Master Thesis No. 478

Simulation of Fluid Flows based on the Data-driven Evolution of Vortex Particles

Vemburaj Chockalingam Yadav
Matriculation No. 4868459

Issued by:	Prof. Dr.-Ing. R. Radespiel Institut für Strömungsmechanik Institutsleiter: Prof. Dr.-Ing. R. Radespiel Technische Universität Braunschweig
Supervisors:	M. Sc. Philipp Holl (TU München) Prof. Nils Thuerey (TU München) Dr.-Ing. Andre Weiner (TU Braunschweig)
Created at:	Physikalisch-basierte Simulationen, Technische Universität München
Publication:	month year

https://doi.org/10.24355/dbbs.084-202103121124-0

https://doi.org/10.24355/dbbs.084-202103121124-0

Technische Universität Braunschweig

Institut für Strömungsmechanik

Master thesis project no. 478
for Vemburaj Chockalingam Yadav
matriculation no. 4868459

Simulation of fluid flows based on the data-driven evolution of vortex particles

1. Introduction

Fluid flow simulations can be approached from two different viewpoints: Lagrangian and Eulerian. Simulations with Lagrangian viewpoint discretize the continuum into discrete sets of particles with fluid properties and simulate the evolution of these particles. On the other hand, simulations with Eulerian viewpoint discretize the domain into a grid and simulate the temporal evolution of the field variables on these sets of spatial grid points. Eulerian simulations of incompressible fluid flow typically involve an expensive pressure solve operation to determine a pressure field that leads to a divergence-free velocity field. Smooth particle hydrodynamics (SPH) approaches enforce mass conservation by design. However, to resolve small scale structures in flows with high Reynolds number, a vast amount of Lagrangian particles is needed. The advantage of vortex particle methods is that the particles carry and preserve vorticity such that particles only exist where where the vorticity field is non-zero [1].

Lagrangian simulations with vortex particles involve discretizing the continuum into sets of particles carrying vorticity. The strength of the vorticity can be modeled as a field surrounding each particle. Such simulations involve only the temporal evolution of the vortex particles with their locations and strengths as state variables. Mass conservation is naturally built into vortex particle methods. Even though these simulations deal with vorticity fields associated with particles, velocity fields can be easily obtained at any point in the domain by summing up the contributions from all vortex particles. However, given the initial velocity field, a corresponding set of vortex particles that represents this initial velocity field needs to be estimated to start the simulation. The particle states corresponding to a given velocity field may be obtained by optimizing random initial states.

https://doi.org/10.24355/dbbs.084-202103121124-0

2. Tasks

The goal of the work during this project is to learn the evolution of parametrized vortex particles using deep neural networks. To achieve this goal, fulfilling the following tasks is required:

1. conduct a literature review on vortex particle methods and deep learning-based approaches for the simulation of fluid flows and related physical phenomena
2. get familiar with the differentiable fluid flow solver *PhiFlow* [2]
3. generate training datasets by running Eulerian simulations with the *PhiFlow* solver
4. select and train neural network models that predict the temporal evolution of vortex particles for a two-dimensional inviscid flow problem in the absence of internal boundaries.
5. investigate possibilities to incorporate internal boundaries in the aforementioned two-dimensional problem and test at least one approach
6. include terms that account for viscous effects in the solution
7. post-process, document, and discuss the obtained results

The progress on the given tasks should be discussed in regular intervals with the supervisors at the ISM and TUM. The supervisor confirms that the resources required to carry out the Master's thesis are available in the intended processing period. This includes, in particular, access to up-to-date literature, required software licenses, and adequate hardware for pre- and post-processing or computing time on suitable servers.

3. Presentation of results

Results have to presented as a clearly structured report supplemented with illustrative diagrams and images. The report should contain a critical discussion of the presented results.

https://doi.org/10.24355/dbbs.084-202103121124-0

.....
(Vemburaj Chockalingam Yadav)

available time: 6 month
handed out on:
handed in on:

.....
Prof. Dr.-Ing. Rolf Radespiel
(first examiner)

.....
Prof. Dr-Ing. Jens Friedrichs
(second examiner)

.....
Dr.-Ing. Andre Weiner
(supervisor ISM)

External supervisors: Prof. Dr. Nils Thuerey, M.Sc. Philipp Holl (both Technische Universität München)

4. Literature

[1] A. Stelle, N. Rasmussen, R.Fedkiw: A Vortex Particle Method for Smoke, Water and Explosions, ACM Transactions on Graphics (2005)

[2] P. Holl, V. Koltun, N. Thuerey: Learning to control PDEs with differentiable physics, ICLR (2020)

[3] P. Holl, V. Koltun, N. Thuerey: Learning to control PDEs with differentiable physics, arXiv preprint arXiv:2001.07457 (2020)

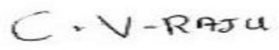
[4] S. Xiong, X. He, Y. Tong, B. Zhu: Neural Vortex Method: From Finite Lagrangian Particles to Infinite Dimensional Eulerian Dynamics. arXiv preprint arXiv:2006.04178 (2020)

https://doi.org/10.24355/dbbs.084-202103121124-0

https://doi.org/10.24355/dbbs.084-202103121124-0

Eidesstattliche Erklärung

Hiermit erkläre ich, Vemburaj Chockalingam Yadav, geb. am 01.10.1994, des Eides statt, die vorliegende Masterrbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet zu haben.



Braunschweig, den 09. Februar, 2021

https://doi.org/10.24355/dbbs.084-202103121124-0

https://doi.org/10.24355/dbbs.084-202103121124-0

Abstract

Fluid solvers that provides accurate and fast fluid simulations are of great importance in many scientific and engineering disciplines. Conventional numerical solvers based on the Eulerian description of the flow provide highly accurate solutions to the Navier-Stokes equations. However, there is typically a significant amount of computational effort is required to execute such Eulerian simulations. On the other hand, fluid solvers built on the Lagrangian description of the flow are more appealing in terms of its vicinity to the true physics, since it treats the actual fluid particles as the primary computational elements. A particular group of Lagrangian particle methods based on vorticity, instead of velocity, as the primary flow variable, delivers velocity field solutions, which are always divergence-free. These vortex methods have an inherent advantage that the particles need to be present only in the regions where vorticity exist, and therefore fewer fluid particles are required to execute simulations as compared to their counterparts with velocity-based formulations.

Recently, deep learning solutions for fluid dynamics problems by the application of artificial neural networks has become more prominent. Neural networks encodes the information about the governing laws of fluid dynamics in its parameters using the knowledge extracted from data samples during training. The aim of this work is to use deep learning to learn the fluid dynamics with Lagrangian vortex particles as the primary flow representation. Solution strategies to train and evaluate the neural networks for predicting Lagrangian vortex particle dynamics for different flow scenarios are presented throughout this work. Conceptualization and implementation of an approach to model interaction between vortex particles based on the Taylor series expansion of the velocity forms the core of this work. We demonstrate that our trained neural networks produce fluid simulations with a reasonable accuracy for different flow scenarios, while respecting appropriate constraints pertaining to fluid dynamics.

https://doi.org/10.24355/dbbs.084-202103121124-0

https://doi.org/10.24355/dbbs.084-202103121124-0

Contents

https://doi.org/10.24355/dbbs.084-202103121124-0

https://doi.org/10.24355/dbbs.084-202103121124-0

Nomenclature

Lateinische Bezeichnungen

D	Domain
L	Length
P	Pressure
p	particle
t	Time
u, v	Velocity components
x, y	Cartesian coordinates
x_p, y_p	Coordinates of particle location

Griechische Bezeichnungen

α	optimization parameter
δ	smoothing constant
ν	kinematic viscosity
θ	Parameters of the neural network
ρ	Dichte
Γ	Circulation
Γ_p	Particle strength
σ_p	Vortex core size of the particle
ω	vorticity

Indizes

BC	Suffix for the input vector to the boundary condition network
G	Grid
div	Divergence
mse	Mean Squared Error
$boundary$	Boundary

Abkürzungen

CFD	Computational Fluid Dynamics
PDE	Partial Differential Equations
ODE	Ordinary Differential Equations
SPH	SmoothedParticle Hydrodynamics
VPM	Vortex Particle Methods

https://doi.org/10.24355/dbbs.084-202103121124-0

https://doi.org/10.24355/dbbs.084-202103121124-0

Chapter 1

Introduction

Fluid simulations have become a major and interesting topic in the field of computer graphics for physics based animations in recent years [9, 17]. Fluids add substantially to the richness of a virtual world due to their ability to assume arbitrary shapes and to show complex behavior [45]. Developing such fluid simulators has always been a challenging task in many applications of computer graphics [7, 9]. A good fluid solver is of great importance in many different areas of scientific and engineering disciplines. In the field of aerodynamics, it is of paramount importance to accurately simulate different flow scenarios for a wide range of Reynolds numbers and Mach numbers and to get a highly accurate and reliable estimates of aerodynamic forces and moments experienced by the whole aircraft [26]. In chemical and process industries, fluid simulations adds substantial value to the underlying transport processes: fluid flow, mixing of fluids, heat transfer, mass transfer and reactions, which is highly crucial for these applications [8]. In the context of computer graphics and special effects industry there is a high demand to convincingly mimic the appearance and behavior of fluids such as smoke, water and fire. This work is focused on computer graphics animation as the driving application, and therefore, we will present all the further arguments under this roof.

Fluid mechanics sits at the heart of the standard mathematical frameworks on which these simulations are based. In fluid mechanics, the entire description of fluid flow is based on a coupled set of highly non-linear partial differential equations called Navier-Stokes equations. Navier-Stokes equations are considered to be the governing equations for fluid flow, the solution of which under different flow scenarios has been expected to provide a complete picture of the flow dynamics. However, like for any non-linear partial differential equations, solving analytically the Navier-Stokes equations is an extremely difficult task and no such closed form analytical solutions exist yet for these equations. In fact, proving for the existence and uniqueness of solution for Navier-Stokes equations is still an open problem.

Numerical methods for solving the Navier-Stokes equations constitutes the branch of Computational Fluid Dynamics (CFD). In simple terms, CFD approaches, like any numerical methods, attempts to convert the system of partial differential equations into a system of algebraic equations, which in turn could be solved using a digital computer. There exists a vast amount of literature on numerical methods for solving Navier-Stokes equations. There exists a wide variety of solvers based on the choice of the desired accuracy, visual appearances and the computing power available. However, in most of the scenarios, the computations involved is too expensive and this increases drastically when the problem in hand demands highly accurate solutions and thus the flow needs to be resolved very fine. High computational complexity of such existing solutions has meant that real-time simulations are difficult to achieve and have been possible only under restricted conditions.

In an ideal situation, the requirements on any fluid solver is that it is highly accurate and

https://doi.org/10.24355/dbbs.084-202103121124-0

provides real-time simulations. Conventional fluid solvers cannot provide both accurate and fast simulations at same time. One needs to sacrifice on the accuracy of simulations using lower flow resolutions, if the priority is to execute simulations very fast and cheap, and vice versa. This is where a new class of fluid solvers based on data-driven learning or simply machine learning, which is becoming more prominent in recent years, comes into the equation.

Unlike classical numerical solvers, which focuses on the numerical solutions of the concerned PDE's to execute simulations, solvers based on machine learning are conceptualized based on learning these laws of fluid mechanics from the knowledge extracted from the underlying sets of fluid data. Once such solvers are trained using the fluid data obtained from classical numerical simulations, they act as a black-box solver to execute further desired simulations. Using such data driven solvers enables the possibilities of real-time simulations, since these solvers would not have to deal with any of the following steps with high computational complexity involved in classical solvers: discretization of the PDE's, solver large linear system of equations multiple times until convergence of the solution, etc.

This master thesis is an attempt on constructing one such fast data driven solver with reasonable accuracy, which simulates the dynamics of fluid flow from a Lagrangian viewpoint by simulating the dynamics of fluid particles carrying vorticity.

Motivation and Scope of the work

When we think about a continuum (like a fluid or a deformable solid) moving, there are two approaches to tracking this motion: the Lagrangian viewpoint and the Eulerian viewpoint.

The Lagrangian approach treats the continuum just like a particle system. Each point in the fluid or solid is labeled as a separate particle, with a position $\mathbf{x}$ and a velocity $\mathbf{u}$. Solids are almost always simulated in a Lagrangian way, with a discrete set of particles usually connected up in a mesh.

The Eulerian approach takes a different viewpoint that is specially used for fluids. Instead of tracking each particle, we instead look at fixed points in space and see how measurements of fluid quantities, such as density, velocity, temperature, etc, at those points change in time. The fluid is probably flowing past those points, contributing one sort of change: for example, as a warm fluid moves past followed by a cold fluid, the temperature at the fixed point in space will decrease, even though the temperature of any individual particle in the fluid is not changing. In addition the fluid variables can be changing in the fluid, contributing the other sort of change that might be measured at a fixed point: for example, the temperature measured at a fixed point in space will decrease as the fluid everywhere cools off.

Numerically, the Lagrangian viewpoint corresponds to a **particle system**, with or without a mesh connecting up the particles, and the Eulerian viewpoint corresponds to using a **fixed grid** that doesn't change in space even as the fluid flows through it.

Grid based methods solves for the incompressible Navier-Stokes equation, written under an Eulerian viewpoint. These methods are one of the most common type in many of the commercially available fluid solvers. In the context of fluid simulations for computer graphics, a typical grid based solver is based on the operator splitting approach [9]. It is nothing but a divide and conquer strategy applied to the differential equations by splitting up a complicated equation into its component parts and then solving each of them separately in turn. The momentum equation in the incompressible Navier-Stokes equation is split up into the advection part, the diffusion part and the pressure/incompressibility part, and solved one after another in sequence. We present in detail the theoretical and mathematical formulations of such a grid based solver in the section 2.2 of Chapter 2. First, we would point out some key difficulties associated with grid based methods.

https://doi.org/10.24355/dbbs.084-202103121124-0

The pressure part of the momentum equation is the final equation to be solved under the operator splitting technique. It is implicitly coupled with the continuity equation of the momentum equation, which implies the velocity field to be divergence-free. This is achieved by solving for a pressure field p that satisfies such a incompressibility constraint. This step is often referred to as *Chorin Projection* [11] and it reduces down to solving the Poisson equation for pressure p. Solving the Poisson equation for pressure p on a discretized grid results in a system of N linear equations of type $\mathbf{Ax} = \mathbf{b}$, where N is the total number of grid cells and matrix $\mathbf{A}$ is of size $N \times N$. The pressure solve step is one of the major bottlenecks in any grid based solvers in terms of the computational complexity involved. This complexity increases rapidly, when finer grids needs to be employed to simulate intricate and complex fluid motions. It hinders the grid based methods from generating real-time fluid simulations.

Another disadvantage that grid based methods suffers from is *numerical diffusion*. The finite difference approximations of the spatial derivatives in the momentum equation introduces errors of the nature that adds to the actual viscosity of the fluid. Thus, for an inviscid flow, the resulting discretized equations would have a numerical viscosity associated with it. These discrete equations are in general more diffusive than the original differential equations, so that the simulated system behaves differently than the intended physical system. Severity of numerical diffusion gets more and more stronger when dealing with coarser grids. Especially, in the context of computer graphics, where one desires to simulate intricate flow motions, Numerical diffusion smears out the flow much faster than it is intended to, and thus it becomes very difficult to simulate intricate flow motions for animation applications in graphics [45].

Furthermore, grid based methods require a computational mesh of the fluid domain. If the geometry of the fluid domain takes complicated shapes, mesh generation often becomes the major bottleneck in the overall simulation process.

Another category of fluid solvers are based on the Lagrangian formulation of the Navier-Stokes equations. Here the basic computational elements are a set of moving fluid particles interacting with each other. These particles carry the associated physical properties of a fluid dynamic system, like mass, density, velocity, vorticity, etc. The trajectories of these particles, as well as the evolution of the transported quantities, are governed by a system of ordinary differential equations (ODE's), the solutions of which provides a description of flow dynamics.

A remarkable feature of particle methods is that their computational structure involves a large number of common abstractions that help in their computational implementation, while at the same time particle methods are distinguished by the fact that they are inherently linked to the physics of the systems that they simulate. In general, particle based approaches are less accurate than their grid based counterparts. This is primarily due to the difficulties in dealing with spatial derivatives on an unstructured particle cloud [28]. However, particle based simulations are typically much easier to program and understand. Furthermore, particle based solvers are much faster, due to the lack of any expensive pressure solve operation, and therefore can be used in real time applications. Also, a salient feature of any Lagrangian particle based methods is that it is mesh-free.

Smoothed Particle Hydrodynamics (SPH) is one of the most common particle based methods, which was first introduced by Monaghan [34] in the context of astrophysics. SPH is based on the primitive velocity formulation of the Navier-Stokes equation. Unlike grid based methods. mass conservation is natural and holds true at all time for any particle based methods, since the particles with their associated masses are tracked during their motion and the mass therefore always remains constant. Even though SPH ensures mass conservation, enforcing the incompressibility constraint and producing divergence-free velocity fields is somewhat difficult [7]. Special treatments usually need to be employed to enforce such constraints in SPH.

A second group of Lagrangian particle simulations based on vortex particles called Vortex Particle

https://doi.org/10.24355/dbbs.084-202103121124-0

Methods are also an interesting proposition [31]. These methods are based on the governing equations with vorticity as the fundamental variable, unlike SPH with velocity. The continuum of the fluid domain to be simulated is discretized into particles carrying vorticity. The strength of the vorticity field of particles can be considered to form a field in space, which could be described by kernels like a gaussian, indicating the variation of the strength of the vorticity around the particle. It is the evolution of this field that the vortex methods deal with and this makes it possible to track every particle in the domain of the fluid flow, i.e., the computation involved in this method is localized.

This Lagrangian description of vorticity particles has many advantages over other conventional CFD techniques such as the finite difference methods and the finite volume methods. The vortex particle method is free of numerical dissipation which is a cause of errors in the grid-based methods. Also, the number of particles is easily adapted to the complexity of the flow. One important characteristic of vortex particle methods is that the velocity field computed from the vorticity field solutions are always divergence free, which is one of the major concerns in SPH. The adoption of the vorticity as the dependent variable avoids the discretization of the irrotational regions while allowing for reserving the computational resources to the vortical zones, which at times can be only a small fraction of the domain, thus providing increased efficiency. This is a major difference with respect to other commonly used computational schemes which require the discretization of the whole solution domain (as for the grid based methods and the Smoothed Particle Hydrodynamics methods).

Massive amounts of data is today widespread across scientific disciplines, and gaining insight and actionable information from them has become a new mode of scientific inquiry as well as a commercial opportunity. Vast amounts of data coupled with advanced computational hardware, efficient data storage and transfer, powerful algorithms and significant investment by industries in data-driven problem solving has fueled the field of machine learning in the past decade [10]. Deep Learning, a class of machine learning algorithms based on deep neural networks [43, 29] has lead to a series of breakthroughs for visual recognition tasks like handwritten digit recognition [30], image classification [29, 21], semantic segmentation [20, 33], object recognition [18, 19, 42], etc and also in the fields of speech recognition [14, 46, 48]. The breakthrough jump in the performances obtained using deep learning in comparison to classical modeling in these fields of computer vision and natural language processing were attributed to the fact that these fields have limited modeling capacity and thus data rich solution using deep learning provides enough scope for data-driven modeling in these fields for wide variety of general scenarios, which are extremely difficult to model using classical approaches. On the other hand, fluid mechanics has a rich literature of physics based models and is rapidly becoming a data rich field [10]. This confluence of first principles and data-driven approaches is unique and has the potential to transform both fluid mechanics and machine learning.

Therefore, in this master thesis we attempt to combine the advantages of both the Lagrangian vortex particle methods and deep learning. We present solution methodologies that uses deep neural networks to learn the underlying dynamics of Lagrangian vortex particles. It is natural to think that one needs to make use of classical numerical solvers based on vortex particle methods to generate dataset for training deep neural networks to learn lagrangian vortex dynamics. However, in this work we resort to execute numerical simulations using a grid based solver. We use *PhiFlow* [24], a fully differentiable PDE solving toolkit, which is written directly using the popular deep learning frameworks of TensorFlow[1] and PyTorch [35]. In this work, we do not explicitlyThe key aspects of this work includes:,

1. generating datasets for training neural network models by running Eulerian simulations using the PhiFlow solver.

2. developing approaches to train deep neural networks that predict the temporal evolution

https://doi.org/10.24355/dbbs.084-202103121124-0

of vortex particles and their associated parameters for two-dimensional inviscid flow in an open domain ,i.e, without any boundaries.

3. developing approaches to incorporate the effects of boundaries for invsicid flows.
4. training the neural networks for flows in presence of viscosity.

https://doi.org/10.24355/dbbs.084-202103121124-0

Chapter 2

Theory: Vortex Methods and Deep Learning

As we have discussed in the previous chapter, there are two different viewpoints for the description of fluid flow: Lagrangian and Eulerian. In simple terms, Lagrangian viewpoint models the flow field and other associated quantities on the basis of tracking motion of fluid particles and their associated properties as they move in time. On the other hand, the Eulerian viewpoint fixates on a particular point in space, and records the properties of the fluid elements passing through that point.

In this Chapter, we will first present the governing equations of fluid flow, both from Lagrangian and Eulerian frame of references. Since, this work is solely based on using Lagrangian vortex particles as the basic computational elements, albeit in a data driven setting, rather than for classical numerical methods, we present a particle based discretized version of the governing equations. We will in detail discuss the governing equations in terms of vorticity discretized by Lagrangian vortex particles, because such a parametrization of flow field by vortex particles forms the core of this work. Since technically, we replace the classical numerical solvers with neural networks as the engine for modeling the particle-particle interactions and simulating the dynamics of such vortex particles, we will present some basics of deep learning, especially the feed forward neural networks, activation functions and backpropagation algorithm, which forms the basis for training any neural network.

2.1 Governing Equations

For convention, we consider the domain of fluid to be D such that $D \subset \mathbb{R}^d$, $d \in \{2, 3\}$, and therefore represent, if any, the internal boundaries of the domain as ∂D. For flows with no internal boundaries, $\partial D = \emptyset$. The boldface letters would represent vector quantities, for e.g. $\mathbf{u}$, represents a velocity vector $\mathbf{u} = [u, v, w]$, with u, v and w representing the components of velocity along the x, y and z coordinate directions respectively. The ∇ operator is the gradient operator, with $\nabla = [\nabla_x, \nabla_y, \nabla_z]$

The Eulerian and Lagrangian description of the mechanics of the fluid flow are related by the *material derivative*. The *material derivative* represents the total time rate of change of any fluid property, for e.g, a scalar field q, for a material element subjected to a velocity field $\mathbf{u}$.

$$\frac{Dq}{Dt} = \frac{\partial q}{\partial t} + \mathbf{u} \cdot \nabla q \tag{2.1}$$

https://doi.org/10.24355/dbbs.084-202103121124-0

The term on the left hand side in Eq. 2.1 represents the total rate of change of a quantity q associated with a fluid particle as it moves through a flow field described by $\mathbf{u}$, which is equal to the sum of the local rate of change (first term in RHS of Eq. 2.1) and convective rate of change of q (second term in RHS of Eq. 2.1).

Solutions to any fluid flow problem are governed by a set of partial differential equations. These equations are referred to as Navier-Stokes Equations, which are derived on the basis of conservation of basic physical quantities like mass, momentum and energy of the fluid. The conservation of momentum could simply realised by setting the quantity q in Eq. 2.1 to velocity $\mathbf{u}$ and balancing the equation with all the possible forces experienced by a fluid element like the forces due to pressure gradient, gravity and viscous forces. From now on, in all the governing equations we would consider only the pressure and viscous forces as the only relevant fluid forces and we would consider only incompressible flows. Eq. 2.2, which represents the equation pertaining to momentum conservation in the Navier-Stokes equations also shows the application of *material derivative* for the transport of momentum. Here, ρ is the fluid density and ν is the kinematic viscosity of the fluid. .

$$\frac{D\mathbf{u}}{Dt} = \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho}\nabla P + \nu \Delta \mathbf{u} \tag{2.2}$$

Mass conservation of fluids under Lagrangian setting is satisfied by the very nature of its definition itself, since we associate individual masses to each of the fluid particles and these masses remains constant throughout the duration for which the motion of these particles are being tracked. However, for incompressible flows under an Eulerian setting, mass conservation would translate to the change in volume of an infinitesimal fluid element around any point in space to be zero under the influence of velocity field. This mathematically translates to the divergence of the velocity field $\mathbf{u}$ being zero, as in Eq. 2.3, which is also referred to as the continuity equation.

$$\nabla \cdot \mathbf{u} = \mathbf{0} \tag{2.3}$$

Equations 2.2 and 2.3 put together forms the complete Navier stokes equations. For a given domain D and a set of suitable boundary conditions, these seemingly simple equations (Eq. 2.4) already form a complete description of incompressible flows with constant viscosity, and it is widely believed that they model complex phenomena such as turbulence and boundary layers correctly. The majority of available flow solvers are based on this formulation in the variables velocity and pressure, which are also called the primitive variables

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} & = -\frac{1}{\rho}\nabla P + \nu \Delta \mathbf{u} \\ \nabla \cdot \mathbf{u} & = \mathbf{0} \end{cases} \quad \text{in } D. \tag{2.4}$$

2.2 Grid Based Methods

Grid based methods solves for the incompressible Navier-Stokes equation, written under an Eulerian viewpoint, given by Eq. 2.5, where $\mathbf{u}$, P, ρ and ν denotes the velocity field, pressure force, fluid density and kinematic viscosity of the fluid respectively.

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} & = -\frac{1}{\rho}\nabla P + \nu \Delta \mathbf{u} \\ \nabla \cdot \mathbf{u} & = \mathbf{0} \end{cases} \quad \text{in } D. \tag{2.5}$$

https://doi.org/10.24355/dbbs.084-202103121124-0

In graphics, well established grid based solvers solves Eq. 2.5 numerically by taking taking advantage of the operator splitting (Chapter 3 of Bridson [9]). It splits the momentum equation in the first line of Eq. 2.5 into the following 3 parts (Eq. 2.6) and solves each of them separately one after another.

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \mathbf{0} \tag{2.6a}$$

$$\frac{\partial \mathbf{u}}{\partial t} = \nu \Delta \mathbf{u} \tag{2.6b}$$

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{1}{\rho} \nabla P \tag{2.6c}$$

$$\text{such that,} \quad \nabla \cdot \mathbf{u} = \mathbf{0} \tag{2.6d}$$

The first equation Eq. 2.6a is purely an advection equation, where the the quantity to be advected by the velocity field $\mathbf{u}$ is the velocity field $\mathbf{u}$ itself. This equation is usually solved by **Semi-Lagrangian** method introduced for computer graphics by Stam [47]. The term Lagrangian stems from the fact that the advection equation Eq. 2.6a, when written from a Lagrangian viewpoint takes the form, $D\mathbf{u}/Dt = \mathbf{0}$ and is utterly trivial that if we were using particle based methods it is solved automatically when the particles are moved through the velocity field $\mathbf{u}$ for a chosen time step of Δt. Therefore, to obtain the velocity at a point $\mathbf{x}_{target}$ at the new time $t + \Delta t$, and to do that in a Lagrangian way, the point $\mathbf{x}_{target}$ is traced back through the velocity field $\mathbf{u}$ over a time Δt. The new velocity at the point $\mathbf{x}_{target}$ is then set to the velocity that the particle, now at $\mathbf{x}_{target}$, had at its previous location $\mathbf{x}_{origin}$ a time Δt ago, Eq. 2.7. If the point $\mathbf{x}_{origin}$ does not correspond to a grid point, a simple interpolation of velocity values at time t form neighbour grid cells is enough.

$$\mathbf{x}_{origin} = \mathbf{x}_{target} - \mathbf{u}_t(\mathbf{x}_{target}), \qquad \mathbf{u}_{t+1}(\mathbf{x}_{target}) = \mathbf{u}_t(\mathbf{x}_{origin}) \tag{2.7}$$

The second equation Eq. 2.6b solves for the effect of viscosity and is equivalent to a diffusion equation. There are many standard approaches that exists for the solution of such type of equations. One straightforward approach is to do a spatial discretization of the diffusion operator applied on the velocity field obtained at the end of advection step and then perform any of the explicit time stepping schemes like Runge-Kutta, as done by Foster and Metaxas [17]. Explicit time integration schemes being not unconditionally stable, implicit time integration schemes are used to solve the diffusion equation by Stam [47]. However, such implicit time stepping involves solving large linear systems of equations, which adds further to the computational cost.

The velocity field obtained after the solving diffusion equation is not divergence free. Since pressure force is simply the force needed to keep the velocity field divergence-free, equations Eq. 2.6c and Eq. 2.6d are solved simultaneously together. This step is often referred to as *Chorin Projection* (Chorin [11]), sicne it projects the intermediate velocity after the diffusion step onto a space of divergence-free velocity field to get the next update of velocity and pressure. It is based on the classical *Helmholtz-Hodge Decomposition* (von Helmholtz [22]) of a velocity field into a divergence free part (solenoidal part) and an irrotational part, Eq. 2.8. The irrotational field $\mathbf{u}_{irrot}$ satsifies $\nabla \times \mathbf{u}_{irrot} = \mathbf{0}$, and thus could be expressed as a gradient of a scalar field q, $(\nabla \times \nabla q = \mathbf{0})$.

$$\mathbf{u} = \mathbf{u}_{sol} + \mathbf{u}_{irrot} = \mathbf{u}_{sol} + \nabla q \tag{2.8}$$

Taking divergence of Eq. 2.8, leads to a Poisson equation for the scalar field q, $\nabla \cdot \mathbf{u} = \Delta q$. The scalar q is nothing but the pressure field p, which could be easily seen if we do a simple

https://doi.org/10.24355/dbbs.084-202103121124-0

discretization of Eq. 2.6c, as shown in Eq. 2.9. Here $\mathbf{u}_{int}$ is the intermediate velocity field, which is the velocity field solution obtained after solving the diffusion equation. Since $\mathbf{u}_{t+1}$ should be divergence-free, $\nabla \cdot \mathbf{u}_{t+1} = 0$, a simple correlation between equations Eq. 2.8 and Eq. 2.9 would imply that $\mathbf{u}$, $\mathbf{u}_{sol}$ and ∇q in Eq. 2.8 corresponds to $\mathbf{u}_{int}$, $\mathbf{u}_{t+1}$ and $\frac{\Delta t}{\rho}\nabla P_{t+1}$ in Eq. 2.9 respectively. Thus, the resulting Poisson equation to solve for the pressure field is given by Eq. 2.10, which could be substituted back to Eq. 2.9 to obtain a divergence free velocity field $\mathbf{u}_{t+1}$. These steps are usually referred to as *pressure solve* or *pressure projection* operations (Chorin [11]).

$$\frac{\mathbf{u}_{t+1} - \mathbf{u}_{int}}{\Delta t} = -\frac{1}{\rho}\nabla P_{t+1} \quad \Longrightarrow \quad \mathbf{u}_{t+1} = \mathbf{u}_{int} - \frac{\Delta t}{\rho}\nabla P_{t+1} \tag{2.9}$$

$$\nabla \cdot \mathbf{u}_{int} = \frac{\Delta t}{\rho}\nabla P_{t+1} \tag{2.10}$$

Solving the Poisson equation Eq. 2.10 for pressure P on a grid results in a system of N linear equations of type $\mathbf{Ax} = \mathbf{b}$, where N is the total number of grid cells and matrix $\mathbf{A}$ is of size $N \times N$. The pressure solve step is one of the major bottlenecks in any grid based solvers in terms of the computational complexity involved. This complexity increases rapidly, when finer grids needs to be employed to simulate intricate and complex fluid motions. It hinders the grid based methods from generating real-time fluid simulations.

2.3 Vortex Particle Methods

As the name suggests, here the formulation of governing equations with vorticity, instead of velocity, as the fundamental variable, forms the basis for describing fluid flow.

A vector field $\mathbf{u}$ that satisfies the incompressibility condition $\nabla \cdot \mathbf{u} = \mathbf{0}$ is called divergence-free and one of the tricky parts of simulating incompressible fluids is making sure that the velocity field stays divergence-free. Pressure force in its simple terms is precisely the force needed to keep the velocity field divergence free. It is evident from Eq. 2.4 that the continuity equation does not contain the pressure variable P, but it shows up in the momentum equation. This means that the pressure is only given implicitly; the continuity equation is a constraint to the momentum equation and the pressure variable acts as a Lagrange multiplier. This leads to significant difficulties in their numerical and theoretical treatment. One therefore could think about the possibilities of eliminating the pressure variable from these equations and this is where vorticity comes to the rescue.

Vorticity field $\boldsymbol{\omega}$ is just the curl of velocity field $\mathbf{u}$ and is given by $\omega = \nabla \times \mathbf{u}$. Intuitively, the vorticity describes the tendency of a fluid particle to rotate around its own centre. The vorticity vector points in the direction of the rotational axis in the three dimensional case, whereas in two-dimensional space the vorticity vector points always in a direction normal to the two-dimensional plane and thus the vorticity values could be considered as a scalar field, which simplifies as $\omega = \partial v/\partial x - \partial u/\partial y$, $\boldsymbol{\omega} = \omega \mathbf{e}_z$, where $\mathbf{e}_z$ is the unit vector in the direction of normal to the 2-D x-y plane.

Taking curl of the momentum equation Eq. 2.2, and with further simplifications like switching of derivatives and using identities of vector calculus, we arrive at the *vorticity equation* Eq. 2.11. For a more detailed step by step derivation of the *vorticity equation*, see [9]. Also, it is evident that the pressure term is absent in Eq. 2.11, since taking curl of a gradient leads to zero ($\nabla \times \nabla P = 0$).

$$\frac{D\boldsymbol{\omega}}{Dt} = \frac{\partial \boldsymbol{\omega}}{\partial t} + \mathbf{u} \cdot \nabla \boldsymbol{\omega} = \boldsymbol{\omega} \cdot \nabla \mathbf{u} + \nu \Delta \boldsymbol{\omega} \tag{2.11}$$

https://doi.org/10.24355/dbbs.084-202103121124-0

The term $\boldsymbol{\omega} \cdot \nabla \mathbf{u}$ in Eq. 2.11 describes*vortex stretching* and forms the core of the description of turbulence energy cascades from large scales to small scales in turbulence. This term exists only for three dimensional flows, whereas for two dimensional flows they reduce to zero, as shown in Eq. 2.12. This makes the three dimensional flows much more complex in comparison to two dimensional ones and it becomes much trickier with the handling of *vortex stretching* terms in numerical simulations. Therefore, dealing with such complex three dimensional flows, especially under a data driven setting, is beyond the scope of this work. We from now on will only present and discuss the vorticity equations and their discretization for two dimensional flows.

$$\boldsymbol{\omega} \cdot \nabla u = (\omega_x, \omega_y, \omega_z) \cdot (\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z}) = (0, 0, \omega_z) \cdot (\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, 0) = 0 \tag{2.12}$$

Thus the complete set of governing equations, which is usually described as the *velocity-vorticity* form of the Navier-Stokes equations, is given for two dimensional incompressible flows by Eq. 2.13.

$$\begin{cases} \frac{\partial \omega}{\partial t} + \mathbf{u} \cdot \nabla \omega = \nu \Delta \omega \\ \nabla \cdot \mathbf{u} = \mathbf{0}, \quad \omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \end{cases} \qquad \text{in } D \subset \mathbb{R}^2. \tag{2.13}$$

The first line of the *velocity-vorticity* formulation Eq. 2.13 is a convection-diffusion equation for the dynamical evolution of vorticity ω under the influence of a known velocity field $\mathbf{u}$.

The second line on the other hand represents the kinematic equation given a known vorticity field ω, and thus the objective is to solve for a velocity field $\mathbf{u}$ with a prescribed curl i.e ω and a prescribed divergence of zero. The solution to the velocity field $\mathbf{u}$, thus could be obtained by solving the resulting Poisson equation Eq. 2.14 [9].

$$\Delta \mathbf{u} = -\nabla \times (\omega \mathbf{e}_z) \tag{2.14}$$

In case of flows with internal boundaries, the Poisson equation is solved subject to a no-through-flow boundary condition at the boundaries. However, for flows with no internal boundaries, i.e, where the domain D is whole of $\mathbb{R}^2$ and the fluid is at rest at infinity, the solution to the velocity field $\mathbf{u}$ could be written as the Biot-Savart integral [31]

$$\mathbf{u}(\mathbf{x}, t) = -\frac{1}{2\pi} \int_D \frac{(\mathbf{x} - \mathbf{y}) \times \omega(\mathbf{y}, t) \mathbf{e_z} \, \mathbf{dy}}{|\mathbf{x} - \mathbf{y}|^2} \tag{2.15}$$

The resulting velocity field $\mathbf{u}$ obtained by Biot-Savart law could also be read as the outcome of convolving the Biot-Svart kernel $\mathbf{K}$ with the vorticity field ω, as shown in Eq.2.16.

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{K} * \omega = \int_D \mathbf{K}(\mathbf{x} - \mathbf{y}) \omega(\mathbf{y}, t) \, \mathbf{dy} \tag{2.16}$$

where $*$ denotes the convolution operation and kernel $\mathbf{K}$ takes the form, as in Eq.

$$\mathbf{K}(\mathbf{x}) = -\frac{1}{2\pi |\mathbf{x}|^2} \begin{pmatrix} x_2 \\ -x_1 \end{pmatrix} \tag{2.17}$$

https://doi.org/10.24355/dbbs.084-202103121124-0

We will also define a term *circulation* Γ of the vorticity field around a closed curve C in 2D, which encloses an area S, and is given by Eq. 2.18. Circulation is a measure of the strength of the vorticity field around any point in the domain.

$$\Gamma = \int_C \mathbf{u} \cdot \mathbf{dx} = \int_S \omega dA \tag{2.18}$$

The main idea behind vortex methods is to approximate the continous vorticity field by a set of discrete vortex particles, each carrying vorticity and the motion of these fluid particles determines the evolution of the vorticity. In Point Vortex Methods, a particle p is a Dirac Delta distribution centred at position $\mathbf{x}_p$ with Γ_p as the particle strengths or the circulation carried by the particles, giving rise to the following approximation, Eq. 2.19.

$$\omega(\mathbf{x}_p(t), t) = \sum_{p=1}^{N_p} \Gamma_p(t)\delta(\mathbf{x} - \mathbf{x}_p(t)) \tag{2.19}$$

where N_p is the number of vortex particles.

By inserting this particle approximation into the Biot-Savart law in Eq. 2.15, one obtains the the resulting velocity field, as in Eq. 2.27.

$$\mathbf{u}(\mathbf{x}, t) = \sum_{p=1}^{N_p} \Gamma_p(t)\mathbf{K}(\mathbf{x} - \mathbf{x}_p(t)) \tag{2.20}$$

However, a particular numerical difficulty with point vortex methods is that the velocity field becomes unbounded if any two vortices come very close to each other, as pointed out in [2]. Also, [6] showed that the computed velocity field is unreliable at locations other than vortex positions.

To handle the above numerical difficulties of the point vortex methods Chorin [12] suggested using *vortex blobs*, instead of point vortices. A vortex blob is obtained by spreading the circulation of a point vortex over a chosen small area, which is referred to as the vortex core. Chorin [12] assumed the core size associated with all the particles to be identical, whereas in the work of Leonard [31], a general formulation was presented wherein the particles were allowed to have different core sizes. This leads to the following approximation of the vorticity field [31], as shown in Eq. 2.21

$$\omega(\mathbf{x}_p(t), t) = \sum_{p=1}^{N_p} \Gamma_p(t)\gamma_p(\mathbf{x} - \mathbf{x}_p(t)) \tag{2.21}$$

where the function γ_p describes the vorticity distribution in the vortex core and satisfies the normalization, Eq. 2.22.

$$\int \gamma_p(\mathbf{x})\mathbf{dx} = 1 \tag{2.22}$$

This function γ_p is also referred to as smoothing function, core function or core shape [31], inherently must contain a parameter σ, which represents the characteristic size of the vortex

https://doi.org/10.24355/dbbs.084-202103121124-0

core. The shape or distribution function γ is common to all vortex particles p, whereas each particle have their own vortex core size parameter σ_p (Eq. 2.23).

$$\gamma_p(\mathbf{x}) = \frac{1}{\sigma_p^2}\phi\left(\frac{\mathbf{x}}{\sigma_p}\right) \tag{2.23}$$

The smoothing function ϕ in Eq. 2.23 is usually chosen to be axisymmetric due to the simplicity involved in evaluating the resulting velocity field. A simple and common choice for such a case would be a gaussian distribution (Eq. 2.24).

$$\gamma_p(\mathbf{x}) = \frac{1}{\pi\sigma_p^2}\exp\left(-\frac{\mathbf{x}^2}{\sigma_p^2}\right) \tag{2.24}$$

There have been several works dealing with the construction of more complex axisymmetric core function with the aim of achieving higher order approximation of vorticity fields and the resulting vortex fields and also from the perspective of convergence of vortex methods using such smoothing functions, with [6, 15, 36, 13] being some of them.

Mathematically, Eq. 2.21 representing the approximation of vorticity field by vortex blobs is equivalent to convolving the point vortex approximation of the vorticity field (Eq. 2.19) with the core function γ. Therefore, the resulting velocity field with vortex blobs takes the form, as given by Eq. 2.25, which resembles very close to its point vortex counterpart (Eq. 2.20).

$$\mathbf{u}(\mathbf{x},t) = \sum_{p=1}^{N_p}\Gamma_p(t)\mathbf{K}_{\sigma_p}(\mathbf{x}-\mathbf{x}_p(t)) \tag{2.25}$$

where $\mathbf{K}_{\sigma_p} = \mathbf{K} * \gamma_p$, which is just the result of the convolution of the Biot-Savart kernel $\mathbf{K}$ (Eq. 2.15) with the core function γ_p. The subscript σ_p in $\mathbf{K}_{\sigma_p}$ indicates that the resulting kernel is parametrized by the vortex core size σ_p.

In case of inviscid flows. the vorticity equation in Eq. 2.13 reduces simply to $D\omega/Dt = 0$. Substituting, either of the point vortex (Eq. 2.20) or the vortex blob (Eq. 2.25) approximation of the vorticity field in the vorticity equation, one obtains Eq. 2.26 (since, we are dealing in a Lagrangian setting, the material derivative simply corresponds to the ordinary time derivative).

$$\frac{d\Gamma_p}{dt} = 0, \quad p = 1,\ldots,N_p \tag{2.26}$$

It is evident that the circulation associated with any vortex particle for two dimensional inviscid flows does not change during its motion, and therefore two dimensional invsicid flows are also referred to as circulation preserving flows. Once a particle is associated with a circulation Γ_p based on the initial vorticity field $\omega(\mathbf{x},0)$, it remains constant during the flow.

The evolution or motion of these particles carriying vorticity is influenced by the local velocity field $\mathbf{u}$, which in turn is obtained from Biot-Savart law (Eq. 2.15) using the point vortex or vortex blob approximation of the vorticity field. Thus, for the case with vortex blobs the governing equations for particle motions are given by, Eq. 2.27, .

https://doi.org/10.24355/dbbs.084-202103121124-0

$$\frac{d\mathbf{x}_p}{dt} = u(\mathbf{x}_p(t), t) = \sum_{q=1}^{N_p} \Gamma_q \mathbf{K}_{\sigma_q}(\mathbf{x}_p(t) - \mathbf{x}_q(t)), \quad p = 1, \ldots, N_p \tag{2.27}$$

Eq. 2.27 represents a coupled system of ordinary differential equations. In principle, these can be solved using any of the classical time integration techniques such as Runge-Kutta or multi step methods.

The situation becomes a bit a bit trickier when dealing with viscous flows, where the vorticity equation in Eq. 2.13 takes the from $D\omega/Dt = \nu\Delta\omega$. Thus, for viscous flows, the flow is no more circulation preserving, rather there is a continous diffusion of vorticity. Conventionally, dealing with diffusion in vortex methods, or in general for any particle based methods, has always been difficult. This has been due to the fact that particle based methods are much more suited and physically sound for partial differential equations involving only convection terms (hyperbolic PDE's), for example the inviscid vorticity equation, which is just a transport equation for vorticity. The viscous vorticity equation brings into the picture the Laplacian / diffusion operator which makes the PDE parabolic overall and therefore adds significant difficulties for the particle based numerical methods.

The vorticity associated with the particles is now a dynamic quantitiy, unlike inviscid flows for which the vorticity associated with any given particle is a constant. There are number of works on vortex particle methods that deals with modeling diffusion for viscous flows. These works focus on the dynamic evolution of the parameters of the vortices: their positions (Random Walk method by Chorin [12]); their vortex core sizes (Core Expansion method by [31]); or their circulations (Deterministic Particle method by Raviart [41]). Each of these approaches present the equations for evolution of vorticity in its own form and presenting and discussing these equations goes beyond the scope of this work.

On the other hand, not dealing with these equations would cause no harm in our work, which is under the data driven perspective. But our work stems from the discretization of the vorticity field using vortex blobs. However, we use a slightly different form, as shown in Eq. 2.28

$$\mathbf{u}(\mathbf{x}, t) = \sum_{p=1}^{N_p} \Gamma_p(t) F(\mathbf{x} - \mathbf{x}_p(t); \mathbf{h}_p(t)) \tag{2.28}$$

where the kernel F, which we would call as *velocity-kernel* and is given by Eq. 2.29

$$F(\mathbf{x}; \mathbf{s}) = \frac{1}{|\mathbf{x}|^2} g(\mathbf{x}; \mathbf{h}) \begin{pmatrix} -x_2 \\ x_1 \end{pmatrix} \tag{2.29}$$

Here, the kernel g would be referred to as the *falloff-kernel*, which models the distribution of vorticity around a particle with vortex strength Γ. One could notice that this *falloff-kernel* is identical to the smoothing function ϕ in Eq.2.23. The falloff-kernels in our case will always be axisymmetric. We parametrize the falloff kernel g, and thus in turn the *velocity kernel* F with the parameter vector $\mathbf{h}$. For a gaussian falloff kernel, the parameter $\mathbf{s}$ would simply be the standard deviation of the gaussian, $\mathbf{h} = [\sigma]$.

Such a parametrization of vorticity field using vortex particles allows us to use deep neural networks to learn and predict the evolution of these parameters $\mathbf{s}_p(t)$ for each particle p along with their respective positions $\mathbf{x}_p(t)$ and vortex strengths $\Gamma_p(t)$. The formulations and discussions of our approaches to achieve such a data driven learning of lagrangian vortex particle dynamics for different flow scenarios forms the core of the next Chapter.

https://doi.org/10.24355/dbbs.084-202103121124-0

2.4 Deep Learning Basics

Deep feed-forward networks, also called feed-forward neural networks, or multi-layer perceptrons (MLP), are the quintessential deep learning models. The goal of a feed-forward network is to approximate some function $f^*(\mathbf{x})$. For example, for a classifier, $\mathbf{y} = f^*(\mathbf{x})$ maps an input $\mathbf{x}$ to a category $\mathbf{y}$. A feed-forward network defines a mapping $\mathbf{y} = f(\mathbf{x};\boldsymbol{\theta})$ and learns the value of the parameters $\boldsymbol{\theta}$ that result in the best function approximation. These models are called feed-forward because information flows through the function being evaluated from $\mathbf{x}$, through the intermediate computations used to define f, and finally to the output $\mathbf{y}$. There are no feedback connections in which outputs of the model are fed back into itself.

Feed-forward neural networks are called networks because they are typically represented by composing together many different functions. The model is associated with a directed acyclic graph describing how the functions are composed together. For example, we might have three functions $f^{(1)}$,$f^{(2)}$ and $f^{(3)}$ connected in a chain, to form $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. These chain structures are the most commonly used structures of neural networks. In this case, $f^{(1)}$ is called the first layer of the network, $f^{(2)}$ is called the second layer, and so on. The overall length of the chain gives the depth of the network. The final layer of a feed-forward network is called the output layer.

During neural network training, we drive $f(\mathbf{x})$ to match the unknown $f^*(\mathbf{x})$. The training data provides us with noisy, approximate examples of $f^*(\mathbf{x})$ evaluated at different training points. Each example $\mathbf{x}$ is accompanied by a label $y \approx f^*(\mathbf{x})$. The training examples specify directly what the output layer must do at each point $\mathbf{x}$; it must produce a value that is close to $\mathbf{y}$. The behavior of the other layers is not directly specified by the training data. The learning algorithm must decide how to use those layers to produce the desired output, but the training data do not say what each individual layer should do. Instead, the learning algorithm must decide how to use these layers to best implement an approximation of f^*, and thus these layers are called hidden layers. The outputs of hidden layers are also sometimes referred to as features.

The functions $f^{(1)}$, $f^{(2)}$ and so on should be non-linear in its inputs when the input output relationship to be learnt is much complex. Most neural networks do so using an affine transformation controlled by learned parameters, followed by a fixed nonlinear function called an activation function.

Fully connected networks are feed-forward networks in which the layers of the network are densely connected. Each layer could be thought of as a non-linear mapping of 1-dimensional vectors form, say $\mathbb{R}^b \to \mathbb{R}^c$. Each component of these vector may be referred to as units and thus in a fully connected network, every unit in one layer is connected to every unit in the next layer. So, in that case, $\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x}) = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$ would be the first layer of the networks that directly acts on input data $\mathbf{x}$, where $\mathbf{W}^{(1)}$ provides the weights of the linear transformation, $\mathbf{b}^{(1)}$ the biases, g is the non-linear activation function and $\mathbf{h}^{(1)}$ the outputs of first layer. Similarly for the function $f^{(2)}$ in the second layer of the network, $h^{(1)}$ would be the input and the outputs would be computed as $\mathbf{h}^{(2)} = f^{(2)}(\mathbf{h}^{(1)}) = g(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$ and son on till the output layer.

If we denote the input $\mathbf{x}$ as $\mathbf{h}^{(0)}$,then for a fully connected feed-forward network with L layers (hidden layers plus output layer), one could write the forward pass of the network as

$$\mathbf{h}^{(l)} = g(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}), \qquad l \in \{1, 2, ..., L\} \tag{2.30}$$

where $\mathbf{h}^{(l)}$ denotes the activations/output of layer l (input layer is the zeroth layer), $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases of the affine transformations at layer l. If $n^{(l)}$ denotes the dimensionality of the activations (or in other words, the number of neurons) of layer l, then $\mathbf{W}^{(l)}$ is a 2D array of dimension $n^{(l)} \times n^{(l-1)}$ and bias $\mathbf{b}^{(l)}$ is a 1D array of size $n^{(l)} \times 1$.

The activation function g is typically chosen to be a function that is applied element-wise. In the

https://doi.org/10.24355/dbbs.084-202103121124-0

earlier works with neural networks, the sigmoid activation function ($g(z) = \sigma(\mathbf{z}) = 1/(1 + e^{-\mathbf{z}})$ is widely used, which takes a real valued input and squashes it between 0 and 1. However, the activations of such a function would usually saturate towards 0 and 1 and thus the gradient at this regions is almost zero. A zero centered counterpart of the sigmoid function is the tangent hyperbolic activation function or *tanh* which takes a real valued input and squashes it between -1 and 1 ($g(\mathbf{z}) = 2\sigma(\mathbf{z})-1$). Similar to sigmoid, *tanh* also suffers from saturation at its extremas and thus leading to vanishing gradient problems [23]. In modern neural networks, common practice is to use the rectified linear unit, or ReLU, defined by the activation function $g(\mathbf{z}) = \max(0, \mathbf{z})$. Here one could see that the gradients for positive activations are always 1, thus no problem of saturation. But the gradients vanishes here too if the inputs to ReLU are negative, but the problem is not so severe as it is for sigmoid or tangent hyperbolic activations. However,this could also tackled by using a leaky ReLU activation which permits small activations for negative input given by

$$g(\mathbf{z}) = \left\{ \begin{array}{ll} \alpha\mathbf{z} & \text{if } \mathbf{z} \leq 0 \\ \mathbf{z} & \text{if } \mathbf{z} > 0 \end{array} \right\} \quad \alpha > 0 \tag{2.31}$$

For networks intended for classification tasks, the output layer should output the prediction probabilities for each class which should sum up to 1. This is usually achieved by using a softmax activation at the output layer, which is given by $g(z_i) = e^{z_i}/\sum_{k=1}^{n^{(L)}} e^{z_k}$ In order to learn the weights $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ at each layer of the network so that the output of the network at last layer for a particular input sample is approximately equal to the desired output, cost function needs to be constructed that penalizes the deviations in network predictions from the desired target. The choice of the cost function usually depends on the task in hand. For regression problems, a simple L^2 error between the model outputs and the true target would be trivial. If $D = \{\mathbf{x}_i, \mathbf{t}_i\}_{i=1}^{N}$ is the dataset from which the training data is sampled from, $\mathbf{o}_i$ is the last layer output of the neural network ($\mathbf{h}^{(l)}$) for the input sample $\mathbf{x}_i$, and $\theta = \left\{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\right\}_{l=1}^{L}$ be the set of parameters to be learned then the cost $J(\theta)$ based on l^2 error is given by

$$J(\theta) = \mathop{\mathbb{E}}_{(\mathbf{x}_i, \mathbf{t}_i) \sim D} \quad \frac{1}{2} \parallel \mathbf{o}_i - \mathbf{t}_i \parallel^2 \tag{2.32}$$

Gradient based learning algorithms are used to learn the parameters $\boldsymbol{\theta}$) in order to minimize these cost functions. It involves computing the gradient of the cost function $J(\boldsymbol{\theta})$) with respect to each parameter in the weight matrix and bias vector of each layer. This is achieved by the back-propagation algorithm first proposed by [43]. Since, a feed-forward neural network is just a composition of functions, computing the gradients of the cost function with respect to the parameters of the network or with respect to the activations at each layer is as simple as applying the chain rule of differentiation. Error backpropagation could be summarized in following steps.

1. Forward propagate the input sample $\mathbf{x}$ ($\mathbf{h}^{(0)}$) to compute the activations at each layer and estimate scalar the cost function $J(\boldsymbol{\theta})$.

2. Compute the derivatives of $J(\boldsymbol{\theta})$) with respect to the last layer outputs $\mathbf{h}^{(L)}$, the exact form of which would depend on the type of cost function chosen, but would be a 1D array of size $n^{(L)} \times 1$. Multiply this vector element-wise the derivatives of the output with respect to the input to the activation function. We will refer to this as term as $\boldsymbol{\delta}^{(L)}$

$$\boldsymbol{\delta}^{(L)} = \frac{\partial J(\boldsymbol{\theta})}{\partial \mathbf{h}^{(L)}} \odot g'(\mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}) \tag{2.33}$$

https://doi.org/10.24355/dbbs.084-202103121124-0

3. Now back-propagate $\boldsymbol{\delta}^{(L)}$ towards the input the input layer to compute $\boldsymbol{\delta}^{(l)}$ for each hidden layer

$$\boldsymbol{\delta}^{(l)} = (\mathbf{W}^{(l+1)^T}\boldsymbol{\delta}^{(l+1)}) \odot g'(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \quad l \in \{L-1, L-2, ..., 2, 1\} \tag{2.34}$$

4. Compute the required gradients using $\boldsymbol{\delta}^{(l)}$ as

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \mathbf{W}^{(l)}} = \boldsymbol{\delta}^{(l)}\mathbf{h}^{(l-1)^T} \quad \frac{\partial J(\boldsymbol{\theta})}{\partial \mathbf{b}^{(l)}} = \boldsymbol{\delta}^{(l)} \quad l \in \{1, 2, ..., L\} \tag{2.35}$$

Once the gradients of the cost function with respect to the parameters are computed, these parameters are updated using a gradient descent type learning method. Usually a mini-batch version of the gradient descent algorithm called stochastic gradient descent (SGD) is employed, which forward passes only a very small subset of the training dataset at a time instead of the whole dataset and updates the paramters based on the errors on this small subset. Then the next sampled subset would be passed to the network, paraneters would be updated and so on. There are several variants of the stochastic gradient algorithm like SGD with momentum ([37]), Adagrad ([16]), RMSprop ([23]), Adam ([27]), etc., which tries to make intelligent updates of the parameters based not only on the current gradient but also on the first and second order running averages of the gradients for each parameters, which helps in faster convergence and lead to better local minimas.

https://doi.org/10.24355/dbbs.084-202103121124-0

Chapter 3

Learning Vortex Particle Dynamics using Neural Networks

In the previous chapters, we have pointed out some of he important aspects of Lagrangian vortex particle methods and also presented the governing equations for such methods. We have also presented some of the fundamentals associated with fully connected networks. Our main goal here is using such deep neural networks to learn and then predict the evolution of Lagrangian vortex particles. In that case, naturally following questions arise: How do we train the neural networks? What type of neural networks do we choose? What goes as input to these neural networks and why? What should the neural network output? What do we compare the neural network outputs to? How do we compute the loss function to train these networks? What are the type of datasets that we create and how do we generate them? How do we take care of appropriate boundary conditions? We present and discuss the answers to all of these questions throughout this chapter.

3.1 Problem Statement

In general, the main objective behind using neural networks as a fluid solver is to advance a state of flow field at some time t to the next time step $t + \Delta t$. The neural network model should learn to encode the underlying laws of flow dynamics. In comparison to classical numerical solvers, which solve for the true physical laws of fluid mechanics given by partial differential equations, neural networks encodes these laws in a parametrized highly non-linear function of the form $\mathbf{y} = \mathbf{f}(\mathbf{x}; \theta)$, where $\mathbf{x}$ and $\mathbf{y}$ could denote the representations for the state of flow at t and $t + \Delta t$ respectively. The parameters θ of this function are not adjusted based on the actual partial differential equations, but are learned from the knowledge of these governing physical laws extracted from actual flow data form numerical simulations.

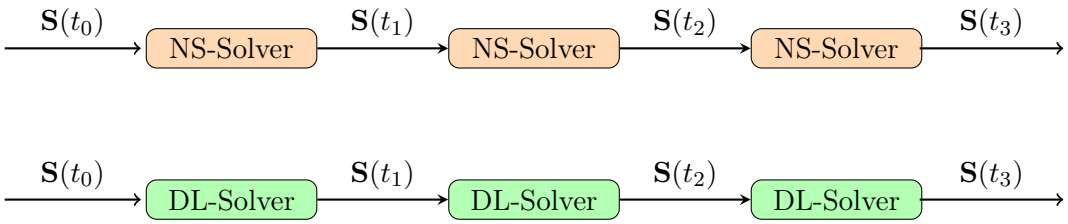


Figure 3.1: General layout of a fluid solver: Numerical Solver (top) and Deep Learning based Solver (bottom).

Let us represent the state of fluid at time t as $\mathbf{S}(t)$. If we represent the physics-based numerical solver and the deep learning based solver by **NS-Solver** and **DL-Solver** respectively, then in

https://doi.org/10.24355/dbbs.084-202103121124-0

very generic terms, we could represent the functionality of these solvers as,

$$\mathbf{S}(t+\Delta t) = \text{NS-Solver}(\mathbf{S}(t)); \qquad \mathbf{S}(t+\Delta t) = \text{DL-Solver}(\mathbf{S}(t)) \tag{3.1}$$

In order to execute simulations over multiple time steps starting from initial time t_0 and a time step of Δt, such that $t_1 = t_0 + \Delta t$, $t_2 = t_1 + \Delta t$ and so on, the solvers are just rolled out in a manner that the output from the solver after one time step, say $\mathbf{S}(t_1)$, goes back as input to the solver to advance the simulation to a next time step t_2 and so on, as shown in Fig. 3.1.

Let the domain of the fluid D be such that $D \subset \mathbb{R}^2$, since we will only be dealing with two-dimensional flows. In the beginning, all the methods pertaining to predicting vortex particle dynamics using deep learning that we present in this work will be for an open domain scenario without any solid boundaries, i.e, $\partial D = \emptyset$. For dealing with flows in presence of boundaries, we will present later an appropriate correction approach based on physics informed neural networks [39]. But for the time being, everything will be presented from the perspective of open domain simulations, unless specified otherwise. In case of open domain, even though the domain is unbounded, one needs to consider only the portion of the domain for flow simulations and further data-driven learning with the assumption that flow goes to rest at infinity. In this work we will consider only square domains of length L, such that $D = [0, L] \times [0, L] := \{(x, y);\ x \in [0, L], y \in [0, L]\}$.

The representation for state of fluid $\mathbf{S}(t)$ to the solvers have different forms depending on the type of the formulations of Navier-Stokes equations the numerical solvers are based upon. In case of grid based solvers, velocity of fluid is the primary variable and thus the representation $\mathbf{S}$ is just the values of the velocity field $\mathbf{u}$ at the discretized locations of the grid cells. For example, if a square domain of length L is discretized by N cells in each of the cooridnate directions and if the velocity values are sampled at centers of this grid cells then we would have the representation $\mathbf{S}$ to be equivalent to the grid based velocity tensor $\mathbf{U}^G$, which would be a 3 dimensional tensor of size $(N \times N \times 2)$, where the last dimension of 2 corresponds to velocity components in each of the coordinate directions. In the context of image processing, such three dimensional tensors are also usually referred to as an $N \times N$ grid with 2 channels. A DL-Solver which uses such a grid based representation usually employs Convolutional Neural Networks [29], since these types of neural networks are tailored to deal with inputs and outputs in the form of a structured 2D or 3D grid.

For particle based methods, the representation S is a set of particles with their associated properties. Thus the Lagrangian representation of the state of fluid $\mathbf{S}$ is given by

$$\mathbf{S} = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \ldots, \mathbf{s}_{N_p}\}, \tag{3.2}$$

where N_p is the total number of particles the fluid is discretized with, and $\mathbf{s}_p$ is the feature/property vector associated with particle p.

$$\omega(\mathbf{x}) = \sum_{p=1}^{N_p} \Gamma_p g(\mathbf{x} - \mathbf{x}_p; \mathbf{h}) \tag{3.3}$$

Our work is based on the grounds of parametrization of the vorticity field and in turn the velocity field by discrete vortex blobs. In this work we will be only dealing with **gaussian vortex particles**. The vorticity field associated with a vortex particle p is given by its vortex strength Γ_p and a falloff-kernel g. The falloff-kernel g models the distribution of vorticity field around the vortex particle. The resultant vorticity at any point in the domain is then just the superposition of vorticities at that point due to all the vortex particles, Eq. 3.3. The gaussian

https://doi.org/10.24355/dbbs.084-202103121124-0

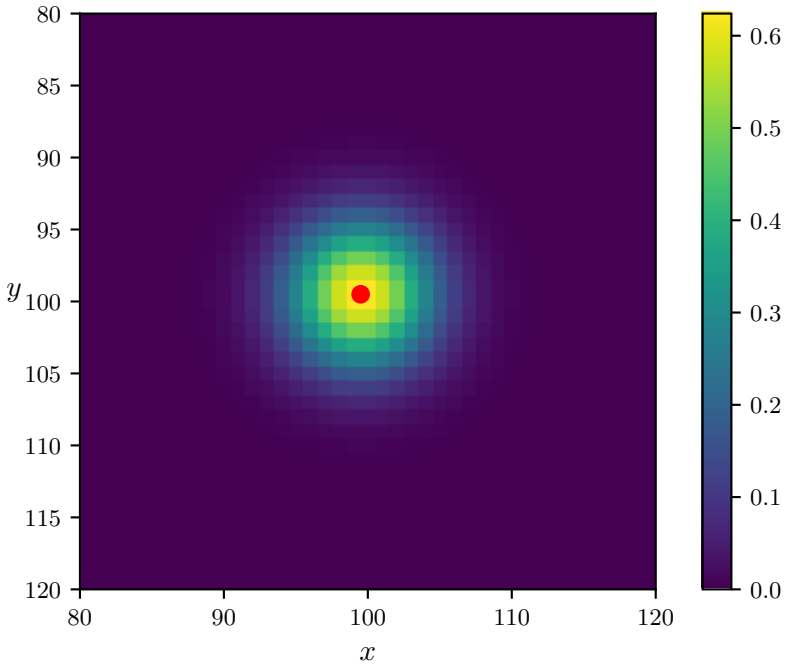


Figure 3.2: Vorticity field created by 1 Gaussian Vortex Particle (shown in red).

falloff kernel g associated with a particle p is

$$g(\mathbf{x};\mathbf{h}_p) = \frac{1}{\pi\sigma_p^2}\exp\left(-\frac{|\mathbf{x}-\mathbf{x_p}|^2}{\sigma_p^2}\right), \tag{3.4}$$

where σ_p is the standard deviation of the gaussian falloff-kernel and in the context of vortex methods, as we presented in the previous chapter, it is referred to as *vortex core size* or simply *core size* of the particle. The gaussian falloff-kernel is centered around the location of the particle $\mathbf{x}_p$ and is parametrized by the parameter vector $\mathbf{h}$, which in this case is just $\mathbf{h}_p = [\sigma_p]$. The resulting vorticity field with the gaussian falloff-kernel could thus be written as

$$\omega(\mathbf{x}) = \sum_{p=1}^{N_p}\frac{\Gamma_p}{\pi\sigma_p^2}\exp\left(-\frac{|\mathbf{x}-\mathbf{x_p}|^2}{\sigma_p^2}\right). \tag{3.5}$$

Using Biot-Savart Law 2.15, it becomes trivial to obtain the velocity field produced as a result of these sets of vortex particles. Substituting Eq. 3.5 into the Biot-Svart integral in Eq. 2.15 and further integrations (for details, refer to [6]), we arrive at the following expression for the velocity field;

$$\mathbf{u}(\mathbf{x}) = \sum_{p=1}^{N_p}\frac{\Gamma_p}{2\pi|\mathbf{x}-\mathbf{x}_p|^2}\left[1-\exp\left(-\frac{|\mathbf{x}-\mathbf{x_p}|^2}{\sigma_p^2}\right)\right]\begin{pmatrix}-(y-y_p)\\ x-x_p\end{pmatrix}. \tag{3.6}$$

Such a parametrization of the velocity field using vortex particles, could be written in generic terms as,

$$\mathbf{u}(\mathbf{x}) = \sum_{p=1}^{N_p}\Gamma_p F(\mathbf{x}-\mathbf{x}_p;\mathbf{h}_p)\frac{1}{|\mathbf{x}-\mathbf{x}_p|}\begin{pmatrix}-(y-y_p)\\ x-x_p\end{pmatrix}, \tag{3.7}$$

https://doi.org/10.24355/dbbs.084-202103121124-0

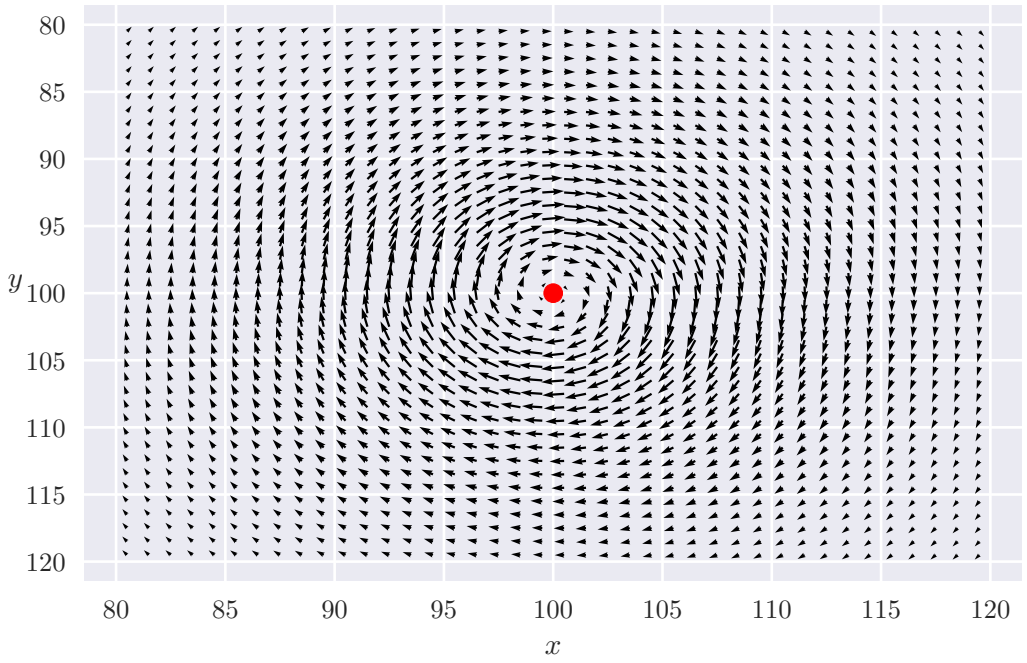


Figure 3.3: Velocity field created by a single vortex particle.

where, we refer to the function F as the velocity-kernel and for a gaussian falloff-kernel, it is given by

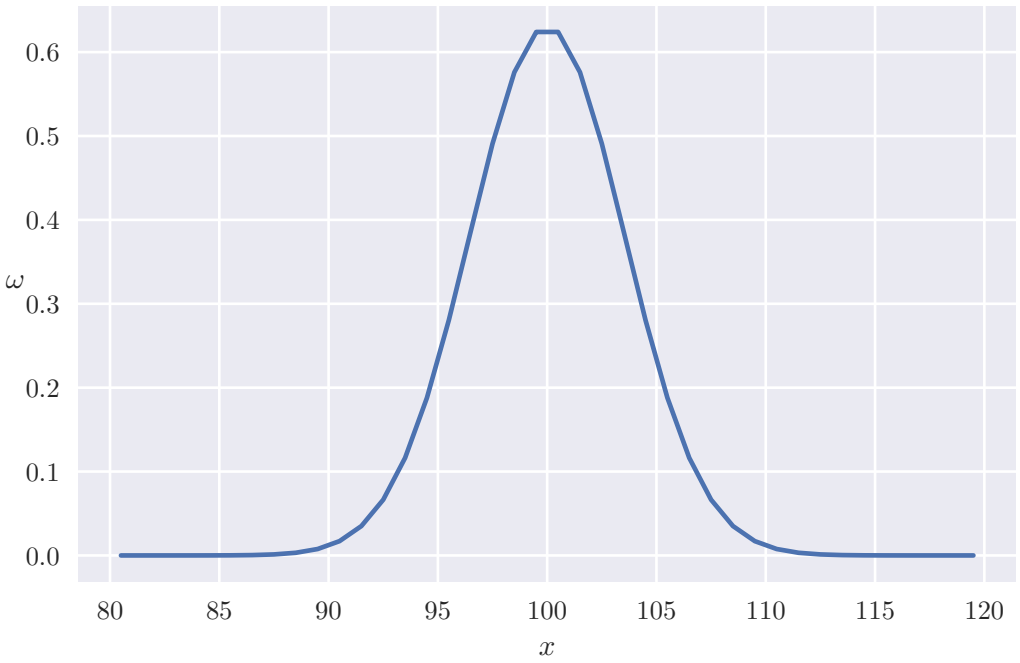
$$F(\mathbf{x};\mathbf{h}_p) = \frac{1}{2\pi|\mathbf{x}-\mathbf{x}_p|}\left[1-\exp\left(-\frac{|\mathbf{x}-\mathbf{x_p}|^{\mathbf{2}}}{\sigma_p^2}\right)\right]. \tag{3.8}$$

The velocity-kernel multiplied by the particle strength gives the magnitude of total velocity at any point as a result of a vortex particle. The resulting velocity vector at any point could just be interpreted as the superposition of velocity vectors at that point due to each of the vortex particles.

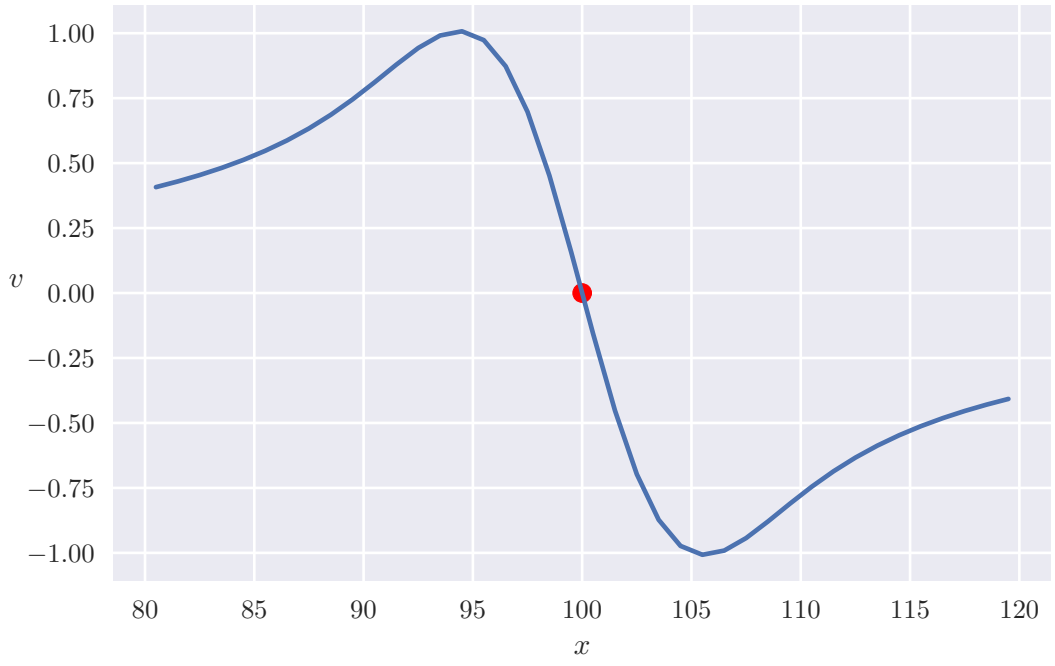
Fig. 3.2 shows the distribution of vorticity field around a single vortex particle. The particle with strength $\Gamma_p = 50$ and a core size of $\sigma_p = 5$ is located at $x_p = 100, y_p = 100$ (shown as red dot). It is evident from Fig. 3.2 that the vorticity field is radially symmetric about its location. Fig. 3.3 shows the velocity vector field as a result of the vorticity induced by such a vortex particle. Also, it is evident that the velocity vector at any point $\mathbf{x}$ points in a direction normal to the radial line connecting the point $\mathbf{x}$ with the particle location $\mathbf{x}_p$. Like the vorticity field, the magnitude of these velocity vectors are also radially symmetric. Fig. 3.4a and Fig. 3.4b shows the variations of vorticity and the y-component of velocity respectively along x-axis plotted at the location of vortex particle. The vorticity is maximum at the particle location and then falls off radially. At any point $\mathbf{x} = (x, y_p)$ located along the horizontal line of vortex particle, the x-component of velocity is zero and thus, the total velocity is just the y-component of velocity. From Fig. 3.4b, we could notice that the magnitude of velocity is zero at particle location, increases radially, reaches a maximum at some point and then falls off to zero. We would get exactly same velocity profile as of Fig. 3.4b if we plotted the variation of x-component of velocity along y-axis at the particle location.

Velocity and vorticity profiles for a single vortex particle, as shown in Fig. 3.4 look much simpler and the shape of these profiles remains the same irrespective of change in the particle strength and core size. However different types and complexities of velocity and vorticity fields could be produced by superposition of number of such gaussian vortex particles each having their own

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) Variation of vorticity field along x-axis.

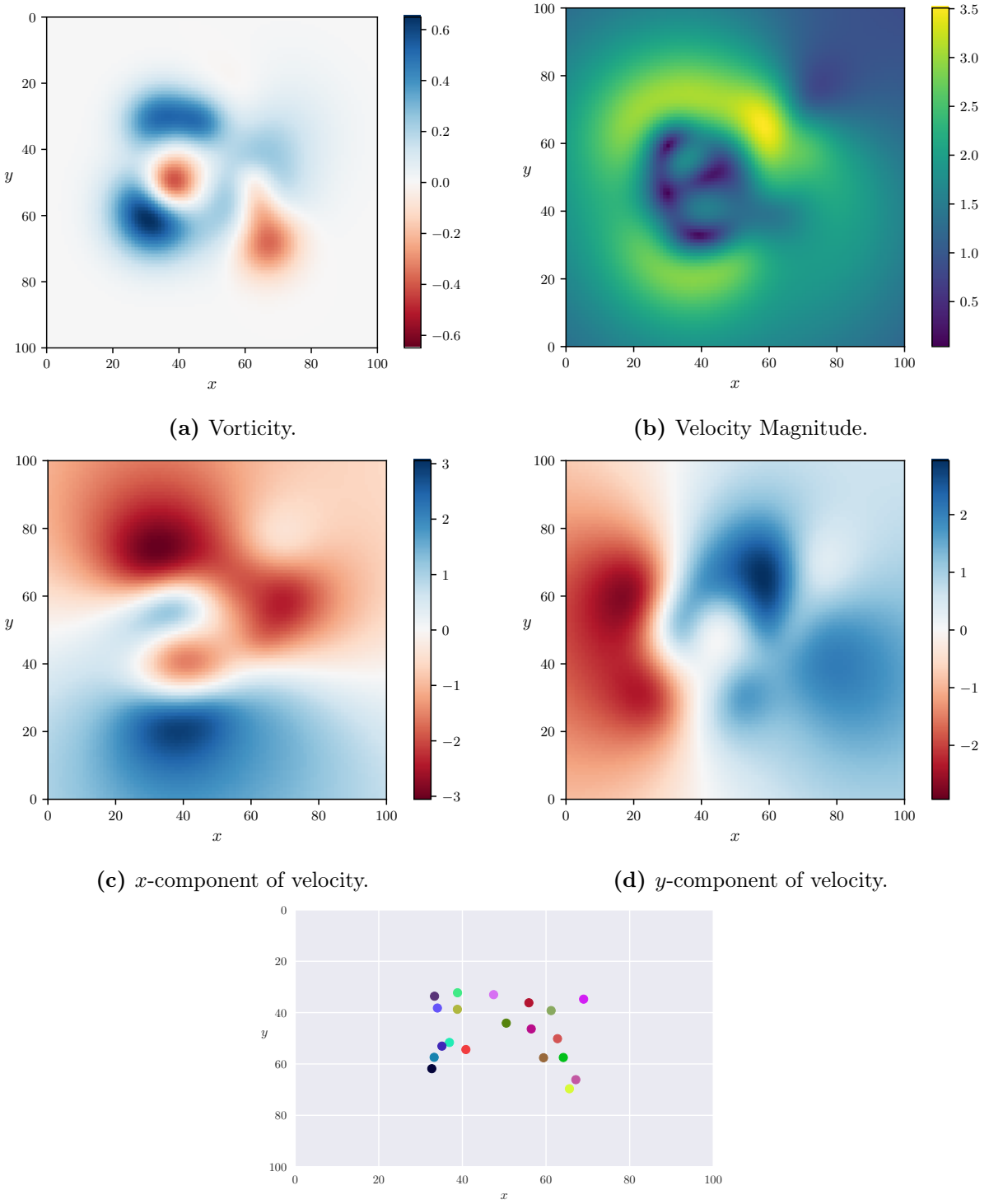


(b) Variation of y-component of velocity along x-axis.

Figure 3.4: Vorticity and velocity profiles of a single vortex particle.

location $\mathbf{x}_p$, vortex strength Γ_p and the vortex core size σ_p. Fig. 3.5 and Fig. 3.7 shows two such examples of parametrization using 20 gaussian vortex particles each on domain of length $L = 100$. The particle locations, their strengths and core sizes are different in both the examples, and thus it becomes possible to produce different shapes and forms of velocity fields (Figs. 3.5c, 3.7c, 3.5d, 3.7d) and vorticity fields (Figs. 3.5a, 3.7a). Also, the velocity at the particle locations

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) Vorticity.

(b) Velocity Magnitude.

(c) x-component of velocity.

(d) y-component of velocity.

(e) Particle locations.

Figure 3.5: Example 1: Parametrization by 20 gaussian vortex particles.

are no more zero unlike the case with single vortex, Fig. 3.6. We visualise the velocity profiles by plotting the variation of y-component of velocity along x-axis at the location of these vortex particles. Fig. 3.6 and Fig. 3.8 shows these plots for the two scenarios corresponding to the examples shown in Fig. 3.5 and Fig. 3.7 respectively. Since the velocity induced by any vortex particle p at its location $\mathbf{x}_p$ is zero, the velocity at $\mathbf{x}_p$ is the resultant of velocities induced by all the other $N_p - 1$ vortex particles at $\mathbf{x}_p$.

https://doi.org/10.24355/dbbs.084-202103121124-0

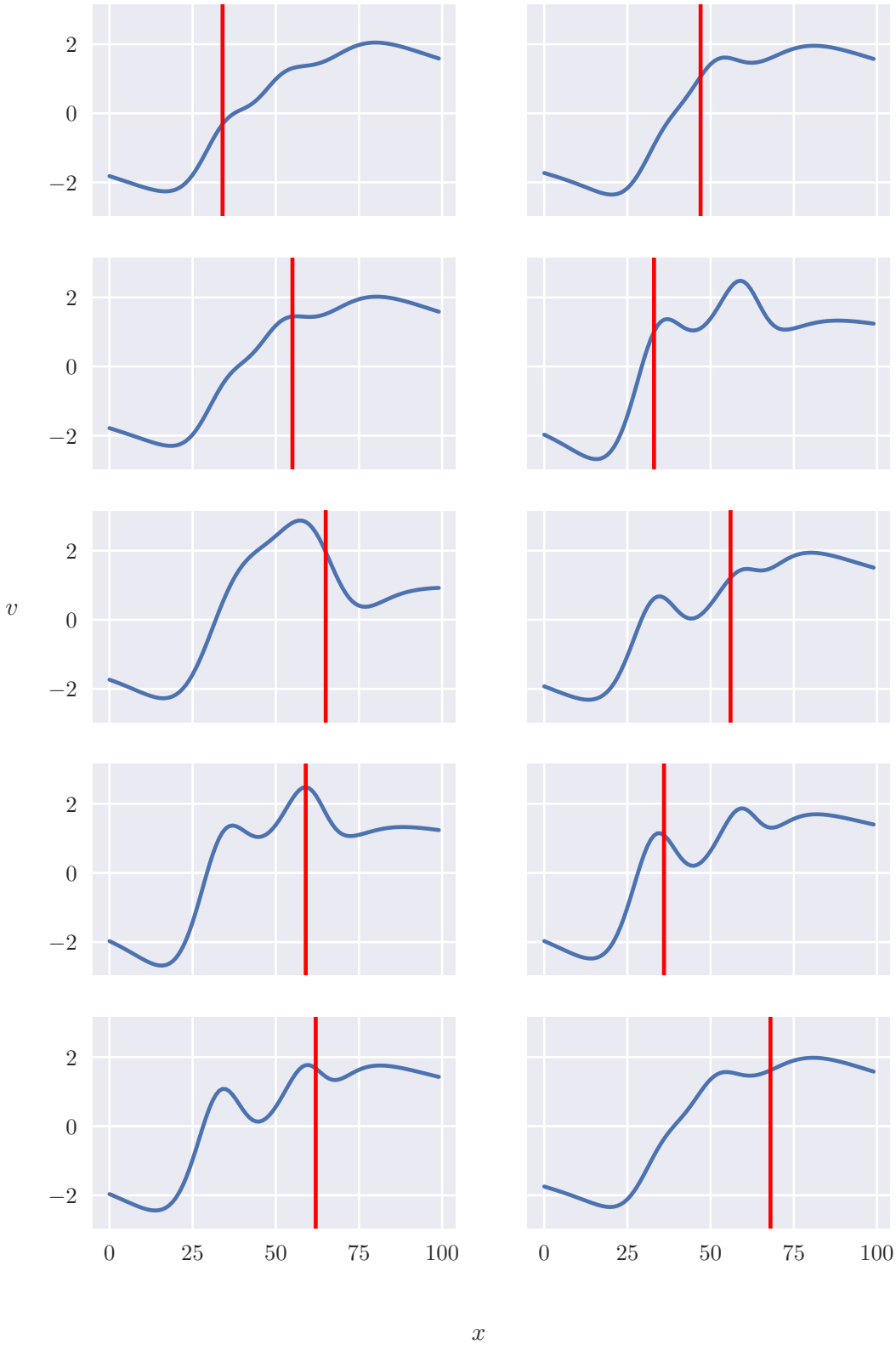
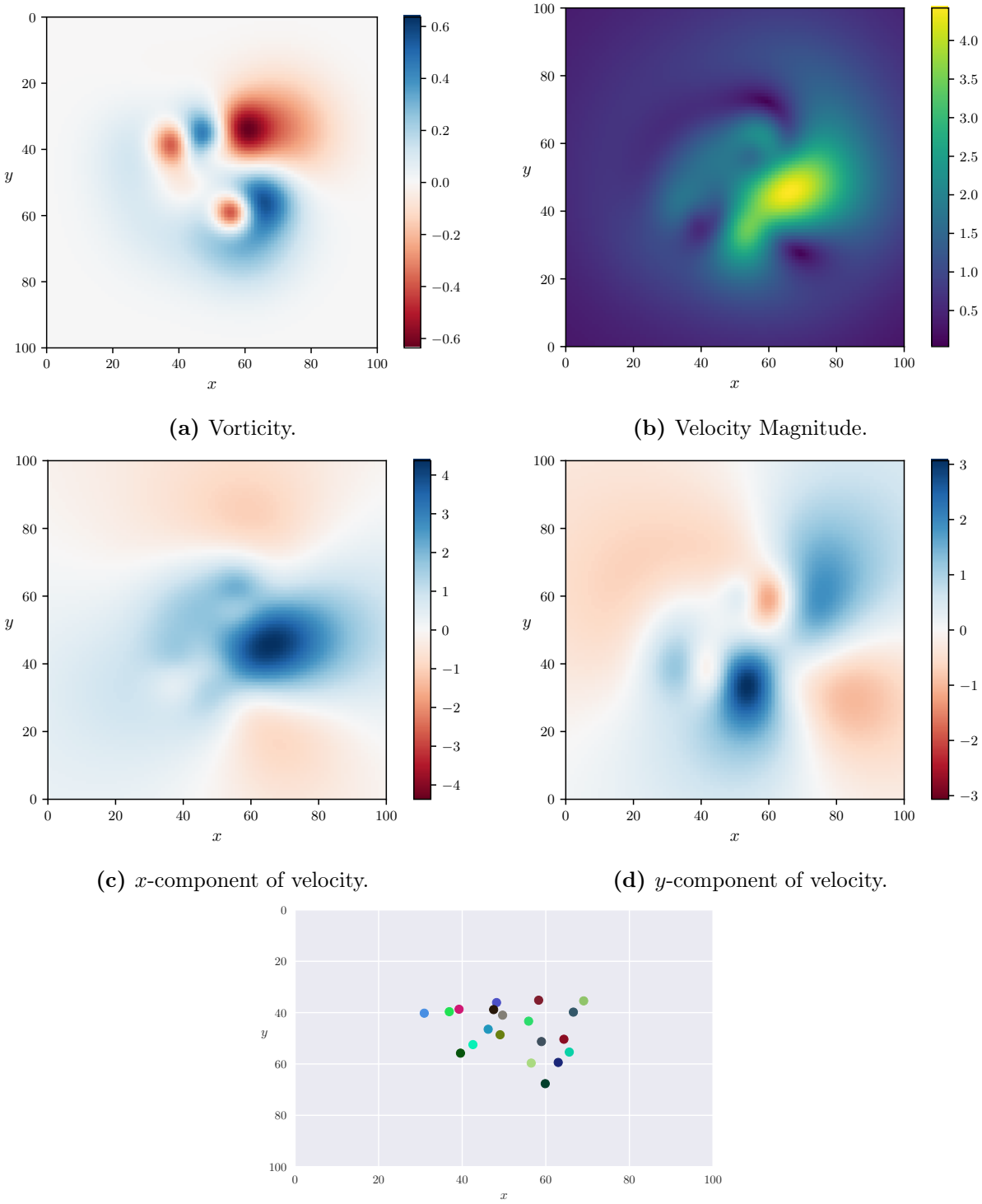


Figure 3.6: Example 1: Variation of y-component of velocity along x-axis plotted at particle locations for 10 of the 20 particles (red line shows the x-coordinate of the particle).

We saw that the the reconstruction of the complete velocity and vorticity fields is possible just by knowing the locations of vortex particles along with their strengths and core sizes. The feature/property vector for any vortex particle would thus include its location, strength and core

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) Vorticity.

(b) Velocity Magnitude.

(c) x-component of velocity.

(d) y-component of velocity.

(e) Particle locations.

Figure 3.7: Example 2: Parametrization by 20 gaussian vortex particles.

size. If the flow under consideration is viscous, we add another element to this feature vector representing the kinematic viscosity ν of the fluid. A set $\mathbf{S}$ of such N_p vortex particles, each with their associated property vector $\mathbf{s}_p$ provides a complete representation of the state of fluid, and

https://doi.org/10.24355/dbbs.084-202103121124-0

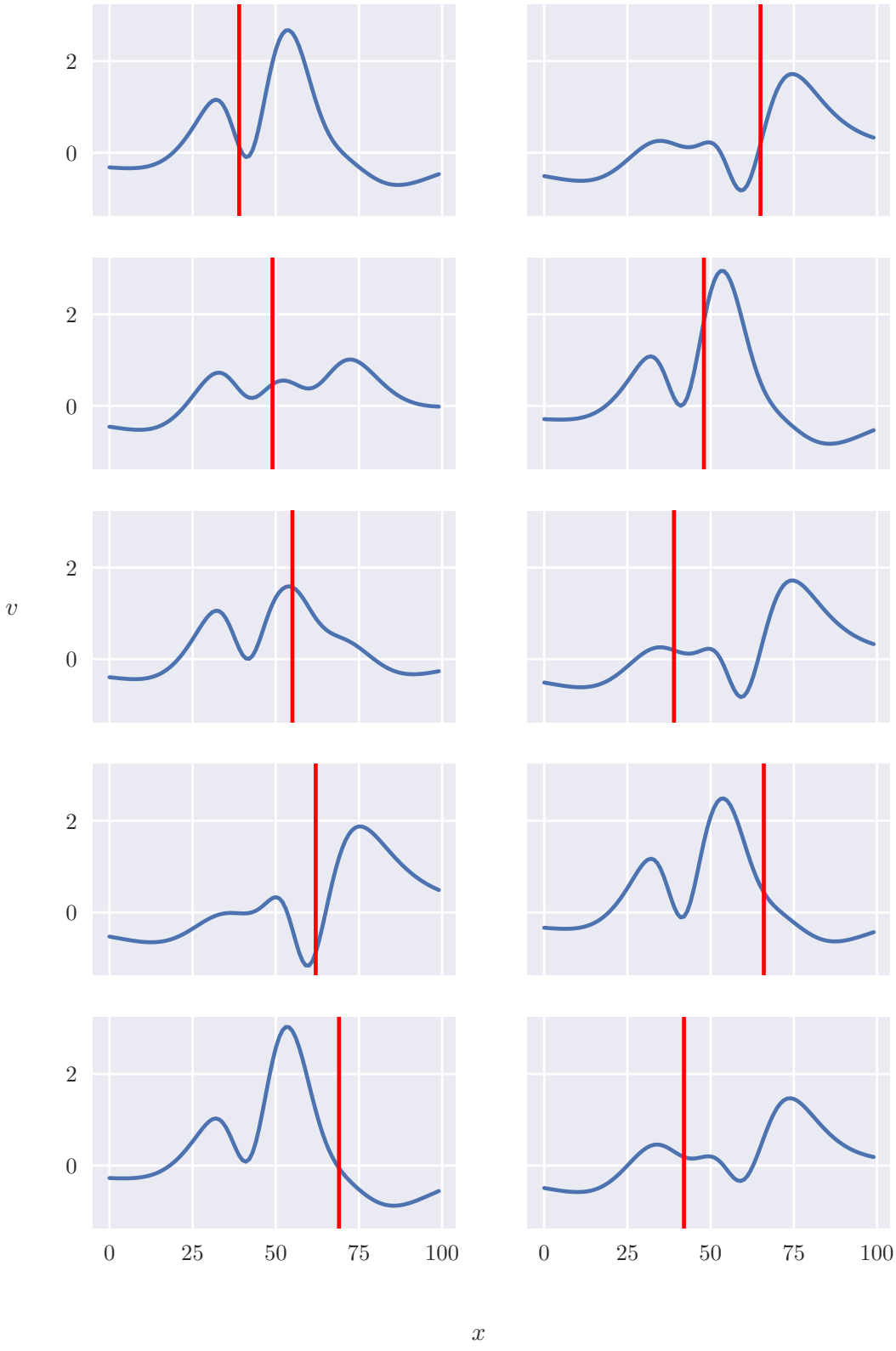


Figure 3.8: Example 2: Variation of y-component of velocity x-axis plotted at particle locations for 10 of the 20 particles (red line shows the x-coordinate of the particle).

is given by

$$\mathbf{S} = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \ldots, \mathbf{s}_{N_p}\}, \quad \mathbf{s}_p = \begin{pmatrix} y_p \\ x_p \\ \Gamma_p \\ \sigma_p \end{pmatrix}, \tag{3.9}$$

https://doi.org/10.24355/dbbs.084-202103121124-0

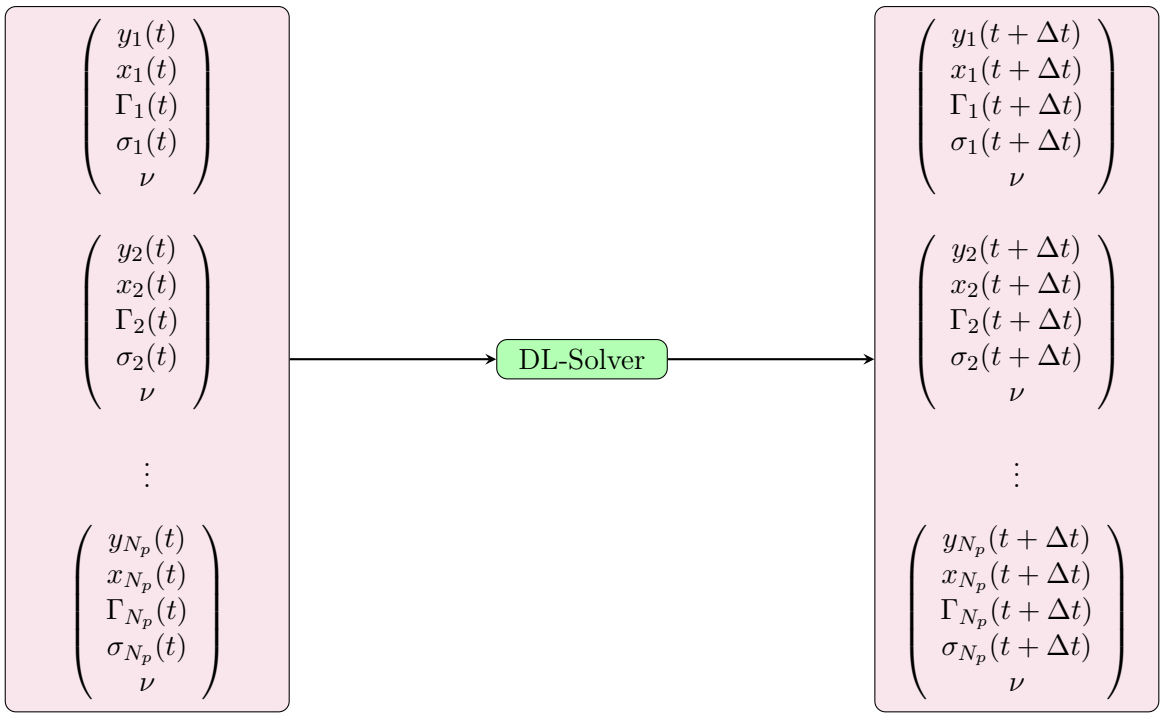


Figure 3.9: Layout of our deep learning based solver for Lagrangian Vortex Dynamics.

and for viscous flows we use the notations as

$$\mathbf{S}^\nu = \{\mathbf{s}_1^\nu, \mathbf{s}_2^\nu, \mathbf{s}_3^\nu, \ldots, \mathbf{s}_{N_p}^\nu\}, \quad \mathbf{s}_p^\nu = \begin{pmatrix} y_p \\ x_p \\ \Gamma_p \\ \sigma_p \\ \nu \end{pmatrix}. \tag{3.10}$$

The functionality of any solver that uses such a Lagrangian vortex particle representation is to predict the evolution of these vortex particles along with their particle strengths and core size. Thus, a deep learning solver that simulates Lagrangian Vortex Dynamics must be capable of taking in and delivering the the representations $\mathbf{S}(t)$ and $\mathbf{S}(t+\Delta t)$ respectively in the form given by Eq. 3.9, as shown in Fig. 3.9.

3.2 Dataset Generation

Neural networks act as a surrogate model of the classical physics based models. Datasets becomes a key asset to train such deep learning models. The task of any neural network is to act as a function approximator that maps input $\mathbf{i}$ to output $\mathbf{o}$ using a paranetrized highly non-linear function of type $\mathbf{o} = f(\mathbf{i}; \theta)$. Neural networks learn to perform these tasks by considering examples form dataset, generally without being programmed with task-specific rules. Neural networks process the data samples to learns the input-output relationships by extracting useful patterns from the data samples and encoding them in its parameters θ.

We have the task of learning the dynamic behaviour of fluids using neural networks. Thus, the data samples that have to be taken into consideration should be generated by executing simulations using conventional numerical solvers that solve for the actual partial differential equations representing the governing laws of fluid dynamics. In our case, the objective is to learn an input-output mapping that maps the gaussian vortex particle representation $\mathbf{S}(t)$ (Eq. 3.9) at time t to its counterpart $\mathbf{S}(t + \Delta t)$ at the next time step. The mapping to be learnt here is the

https://doi.org/10.24355/dbbs.084-202103121124-0

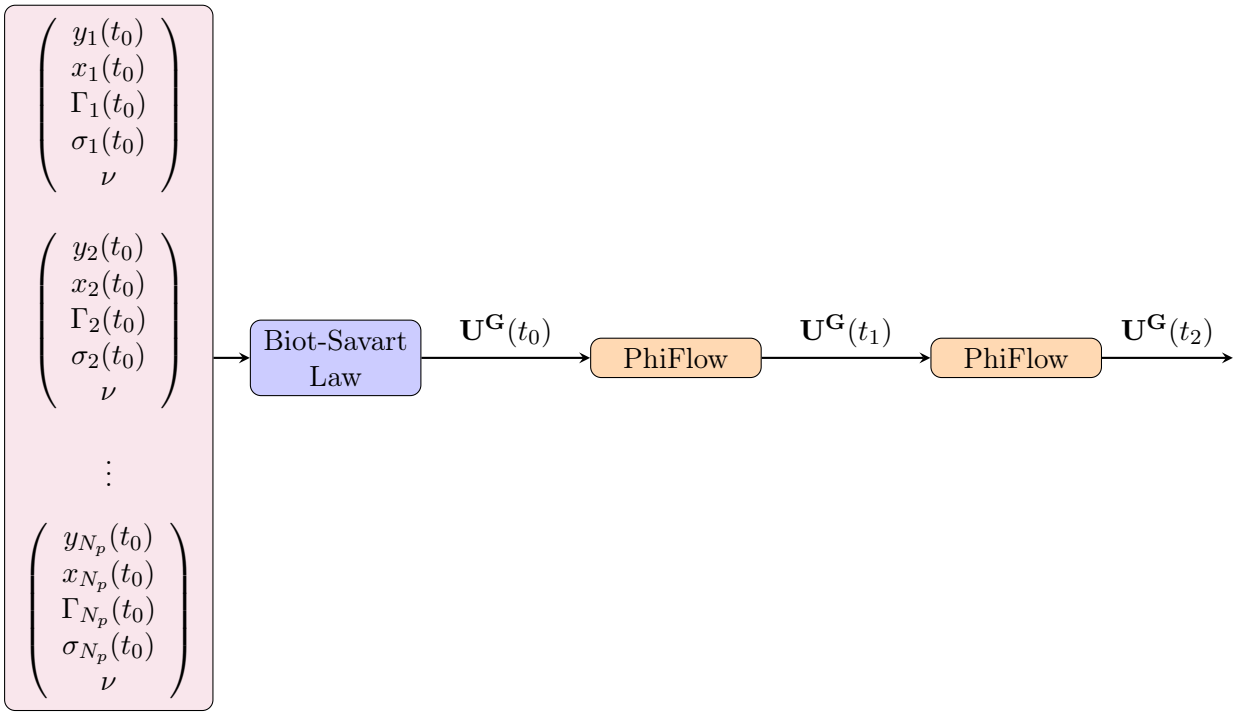


Figure 3.10: An example of data sample generation using *PhiFlow*.

dynamics of Lagrangian vortex particles over a time step Δt which are based on the governing laws corresponding to the *velocity-vorticity* formulation of Navier-Stokes equation Eq. 2.13. For supervised learning of any task using neural networks, the dataset is usually comprised of sets of input-output pairs representing the type of function to be learned. So, in an ideal scenario, we should have a dataset of the following type

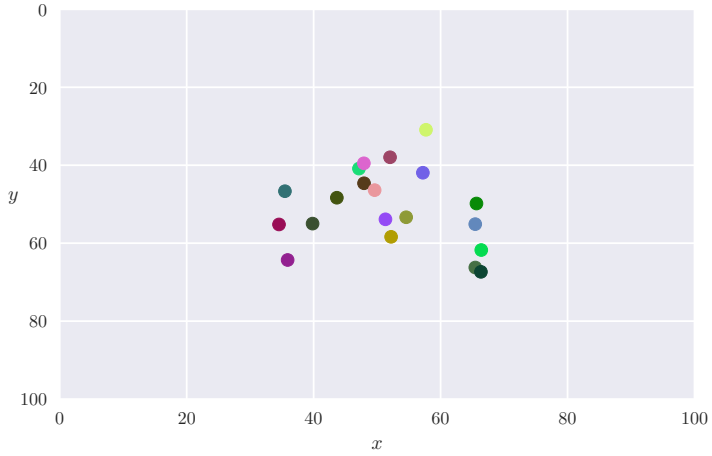
$$\mathbf{D}_s = \left\{ \left(\mathbf{S}_i(t), \mathbf{S}_i(t + \Delta t) \right)_{i=1}^{N_s} \right\}, \tag{3.11}$$

where N_s is the total number of data samples. Here $\mathbf{S}_i(t)$ and $\mathbf{S}_i(t + \Delta t)$ represents the sets of feature vectors of vortex particles at time t and $t + \Delta t$ respectively for a particular data sample. In order to create such data samples, the need would be to use numerical solvers based on Vortex Particle Methods. However, in this work, we will be using the grid based solver for the Navier-Stokes equations from *PhiFlow* [24] instead of a solver based on vortex particle methods for executing numerical simulations. *PhiFlow* is an open-source fully differentiable PDE solving toolkit in Python and we use it due to the simplicity involved in running numerical simulations. One could use the dataset generated by a grid-based solver for training neural networks which learns to predict the vortex particle dynamics in an open domain scenario, due to the explicit mapping from vorticity field to velocity field provided by the Biot-Savart Law.

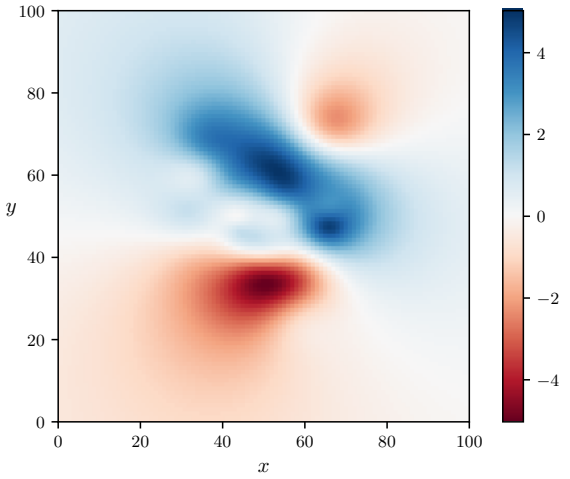
PhiFlow is based on the grid based representation of velocity field $\mathbf{U^G}$. In order to execute simulations using *PhiFlow*, we need a grid based velocity field $\mathbf{U^G}(t_0)$ at some initial time t_0 to start the simulations, which in turn yields the velocity fields $\mathbf{U^G}(t_1)$, $\mathbf{U^G}(t_2)$, $\mathbf{U^G}(t_3)$ and so on at future time instants. Given that we have a vortex particle representation of the fluid $\mathbf{S}(t_0)$ at initial time t_0, the need would be to map it first to a grid based velocity field $\mathbf{U^G}(t_0)$. Using Biot-Savart Law Eq. 2.15, we have an expression for computing the velocity at any point in the domain given the vorticity field. In case of vorticity fields being parametrized by discrete gaussian vortex particles, we have already presented such expression for velocity field computations Eq. 3.6.

Fig. 3.10 shows the layout for creating one data sample using *PhiFlow*. For a domain of length L and say we take N_p number of particles, we first sample N_p random locations, strengths and core

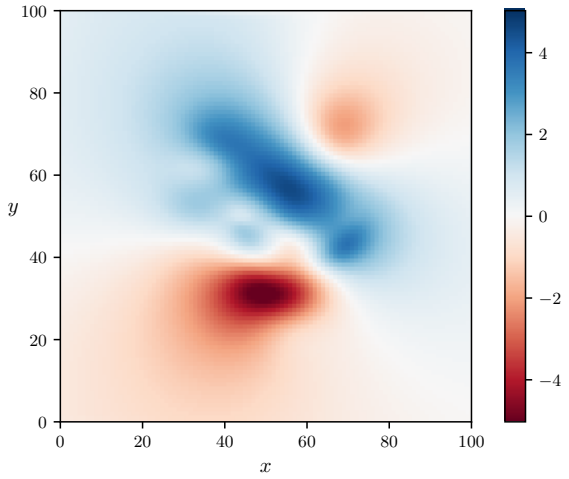
https://doi.org/10.24355/dbbs.084-202103121124-0



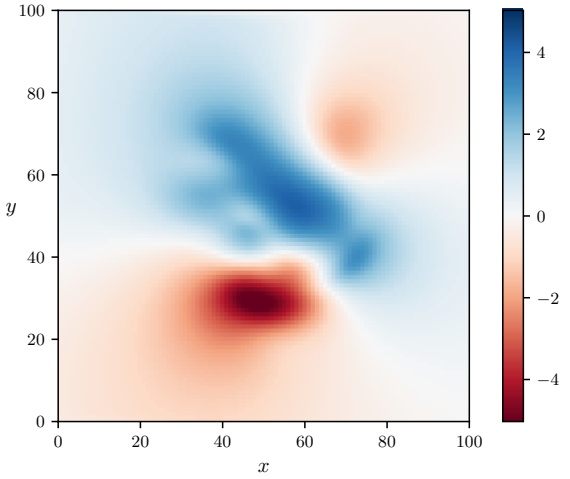
(a) Particle Locations at $t = 0s$.



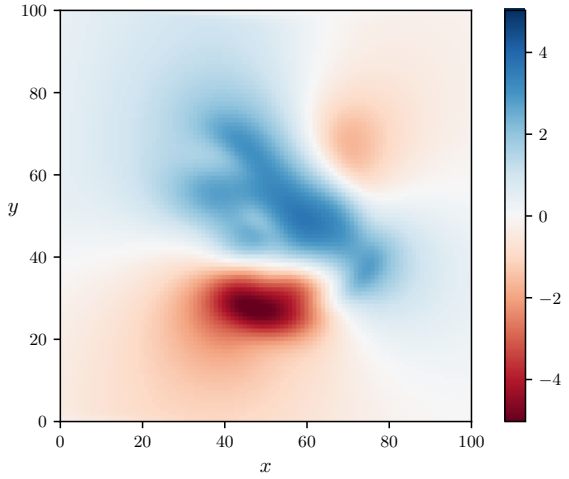
(b) x-velocity at $t = 0s$.



(c) x-velocity at $t = 1s$.



(d) x-velocity at $t = 2s$.



(e) x-velocity at $t = 3s$.

Figure 3.11: An example of grid based velocity fields produced by numerical simulation from *PhiFlow*.

sizes. This gives us the vortex particle representation $\mathbf{S}(t_0)$. Say, we consider the discretization of domain using N grid cells in each direction. We evaluate the velocity values at the locations corresponding to the center of this grid cells using Eq. 3.6 and therefore arrive at the grid

https://doi.org/10.24355/dbbs.084-202103121124-0

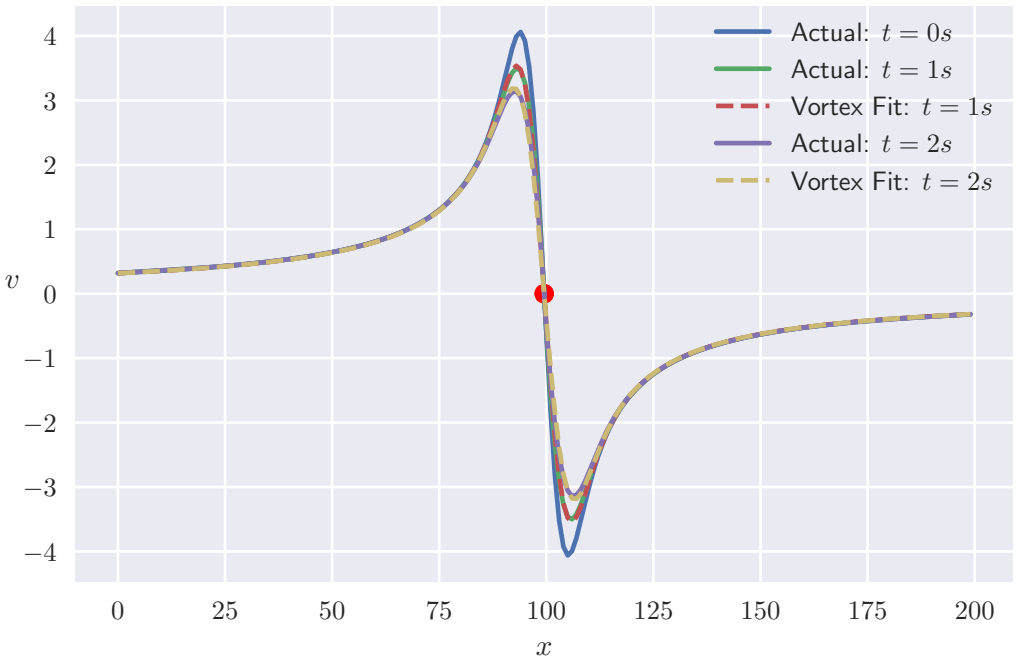


Figure 3.12: Comparison of actual velocity profiles at different time instants with the velocity curves obtained using Vortex-Fit for a single particle case.

based velocity field $\mathbf{U}^{\mathbf{G}}(t_0)$. The mapping form a particle based representation to a grid based representation could be written in generic terms as

$$\mathbf{U}^{\mathbf{G}}(t) = \text{Biot-Savart}(\mathbf{S}(t)). \tag{3.12}$$

We provide $\mathbf{U}^{\mathbf{G}}(t_0)$ as the initial velocity field to *PhiFlow* and advance the simulation from thereon to get the grid based velocity fields at future time steps, as shown in Fig. 3.10. We therefore have a dataset of the following type:

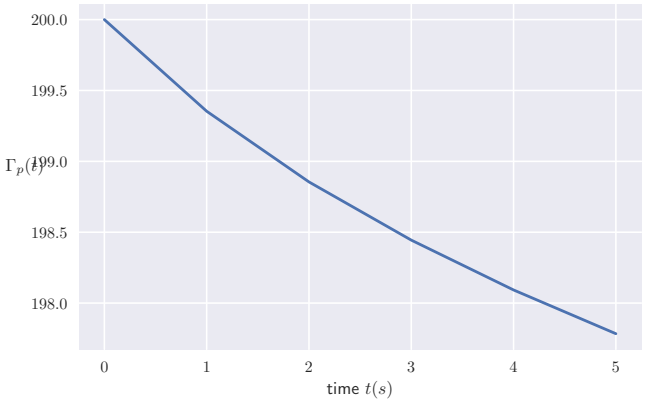
$$\mathbf{D}_s = \left\{ \left(\mathbf{S}_i(t_0), \mathbf{U}_i^{\mathbf{G}}(t_1), \mathbf{U}_i^{\mathbf{G}}(t_2), \dots \right)_{i=1}^{N_s} \right\}. \tag{3.13}$$

Fig. 3.11 shows an example of velocity fields generated using *PhiFlow* simulations. The initial velocity field Fig. 3.11b is obtained for a vortex particle configuration Fig. 3.11a at $t = 0s$ using Eq. 3.6, *PhiFlow* advances this initial velocity field (Fig. 3.11b) to velocity fields at $t = 1s$ (Fig. 3.11c), $t = 1s$ (Fig. 3.11d), $t = 3s$ (Fig. 3.11e), and so on.

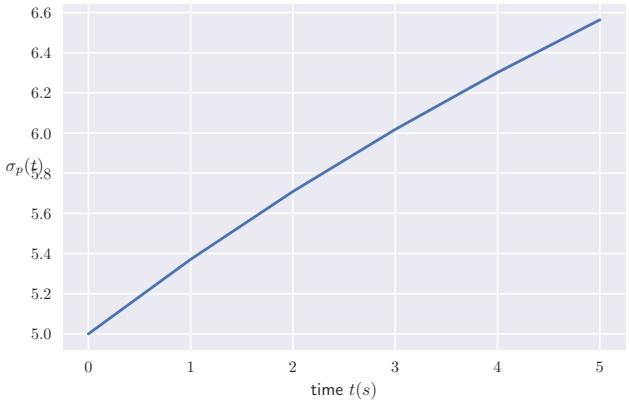
It is clear that we don't directly have the vortex particle representation $\mathbf{S}(t+\Delta t)$ to penalise the neural network predictions during training. However, we could compute the resulting grid based velocity field from the vortex particle representations delivered by our deep learning model using Eq. 3.6. It could be then compared with the true velocity field $\mathbf{U}^{\mathbf{G}}(t+\Delta t)$, which we have from our dataset. We would present the exact formulations of such a loss function for training our neural networks later in this Chapter.

We have the luxury of having both the vortex particle representation $\mathbf{S}$ and the corresponding grid based representation $\mathbf{U}^{\mathbf{G}}$ only for the initial time t_0. For future time steps, only the grid based velocity fields would be available from numerical simulations. From t_0 to t_1, the particles would now have moved form $\mathbf{x}_p(t_0)$ to a new location $\mathbf{x}_p(t_1)$ and also their strengths and core sizes would also have changed to $\Gamma_p(t_1)$ and $\sigma_p(t_1)$ respectively. We have only the convenience

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) Vortex strength $\Gamma_p(t)$.



(b) Vortex Core size $\sigma_p(t)$.

Figure 3.13: Evolution of particle strength and core size obtained by Vortex-Fit for a single particle case.

of computing velocity field based on sets of vortex particles, Eq. 3.6. However, going the other way round is non-trivial. Given a grid based velocity field, we need to obtain the vortex particle representation with sets of vortex particles and their feature vectors, which would have possibly produced the given velocity field. There is a simple trick to tackle this problem.

Since we need to obtain a vortex particle representation $\mathbf{S}$ that best fits a given grid based representation $\mathbf{U}^{\mathbf{G}}$, this could be achieved by solving an optimization problem. We know that the vortex particle representation is a set of vortex particles with their feature vectors, where feature vectors of each particle has its location, strength and core size Eq. 3.9. Thus, the objective would be to find the optimal location, strength and core size of each particle that best fits the given velocity field. Thus, the parameters for optimization are now feature vectors of each particle and therefore we just denote the parametrized vortex particle representation as $\widetilde{\mathbf{S}}(t;\boldsymbol{\beta})$, where $\boldsymbol{\beta}$ is the vector with locations, strengths and core sizes of all particles. We therefore also denote the resulting parametirzed grid based velocity field as $\widetilde{\mathbf{U}}^{\mathbf{G}}(t;\boldsymbol{\beta})$ We could therefore, write the optimization problem as

$$\boldsymbol{\alpha} = \arg\min_{\boldsymbol{\beta}} \quad \left\| \mathbf{U}^{\mathbf{G}}(t) - \widetilde{\mathbf{U}}^{\mathbf{G}}(t;\boldsymbol{\beta}) \right\|^2 \tag{3.14}$$

$$= \arg\min_{\boldsymbol{\beta}} \quad \left\| \mathbf{U}^{\mathbf{G}}(t) - \text{Biot-Savart}(\widetilde{S}(t;\boldsymbol{\beta}) \right\|^2 . \tag{3.15}$$

https://doi.org/10.24355/dbbs.084-202103121124-0

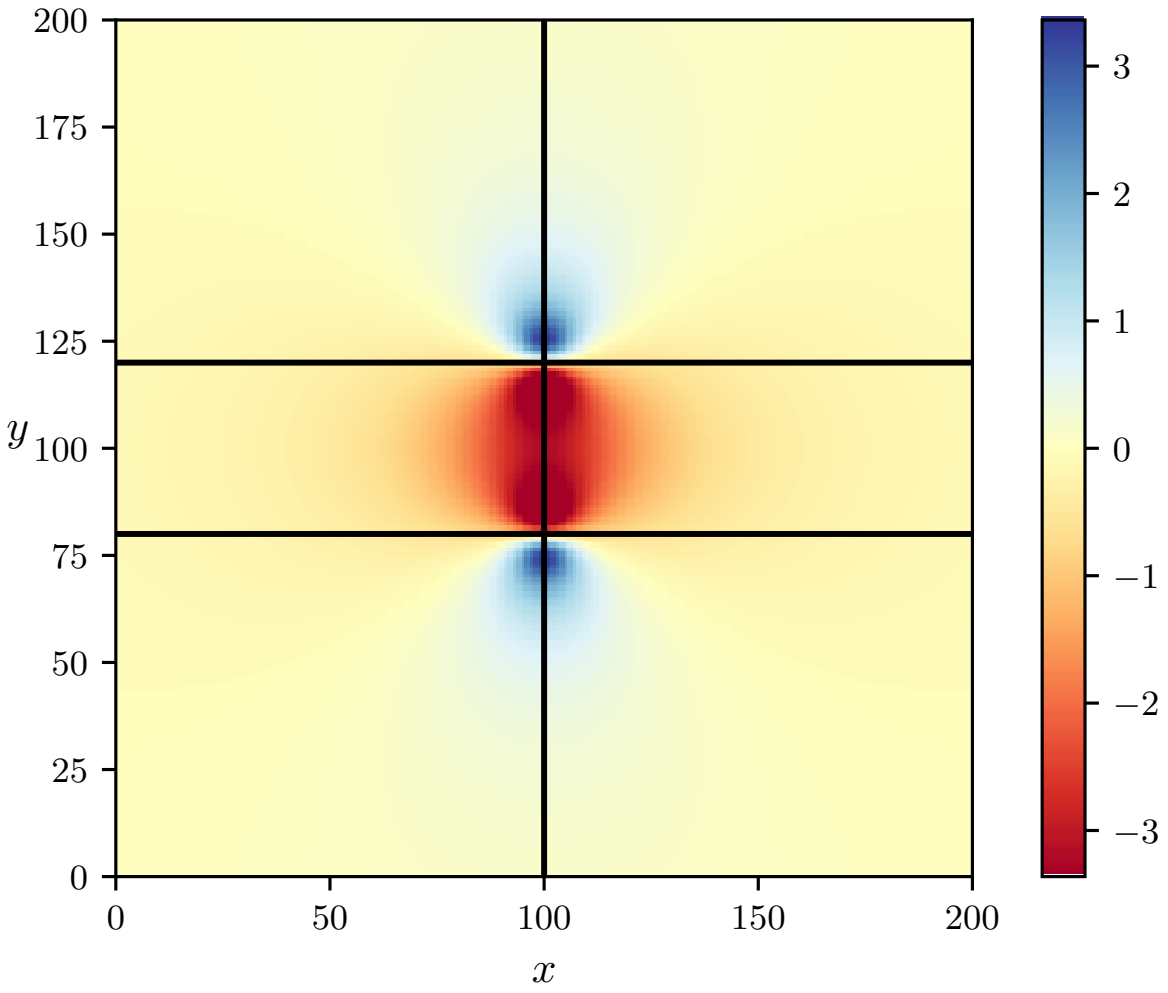


Figure 3.14: Velocity field (x-component) at $t = 0s$ for 2 particles located along the same vertical line, having strengths of same magnitude but opposite sign and having same core sizes. (Intersection of black lines correspond to particle locations).

Thus $\boldsymbol{\alpha}$ is the final state of parameter $\boldsymbol{\beta}$ after optimization, which contains the optimal features of each particle. However, the optimization problem represented by Eq. 3.15 is highly non-convex. This could be easily justified by the fact that the particle locations and core sizes are inside an exponential function for gaussian vortex particles. Any objective function which leads to non-convex optimization has more than one local minima. Thus, we could potentially have multiple sets of particle feature vectors that could best represent the given velocity field. However, say we want to get the vortex particle representation for velocity field $\mathbf{U}^{\mathbf{G}}(t_1)$ at time t_1, and say we have the the actual vortex particle representation $\mathbf{S}(t_0)$ at some previous time t_0, such that $t_1 - t_0 = t$ and Δt is small. Then, initializing the optimization parameter $\boldsymbol{\beta}$ in Eq. 3.15 with the the particle feature vectors at time t_0 and then solving the optimization problem would result in a solution, which is physically consistent with the particle dynamics from t_0 to t_1.

We refer to the process of obtaining such particle based representation for a given velocity field using optimization as **Vortex-Fit**. We present an example of such a Vortex-Fit operation for a single vortex particle in Fig. 3.12. We perform Vortex-Fit on the actual velocity fields at times upto $t = 5s$ with a step of $1s$. The velocity curves obtained by Vortex-Fit fits well in comparison to the actual velocity profiles from numerical simulations. Fig. 3.13 shows the particle strengths and core sizes obtained by Vortex-Fit for the case shown in Fig. 3.12. The evolution curves in Fig. 3.13a and Fig. 3.13b are physically consistent with the actual dynamics of a gaussian vortex

https://doi.org/10.24355/dbbs.084-202103121124-0

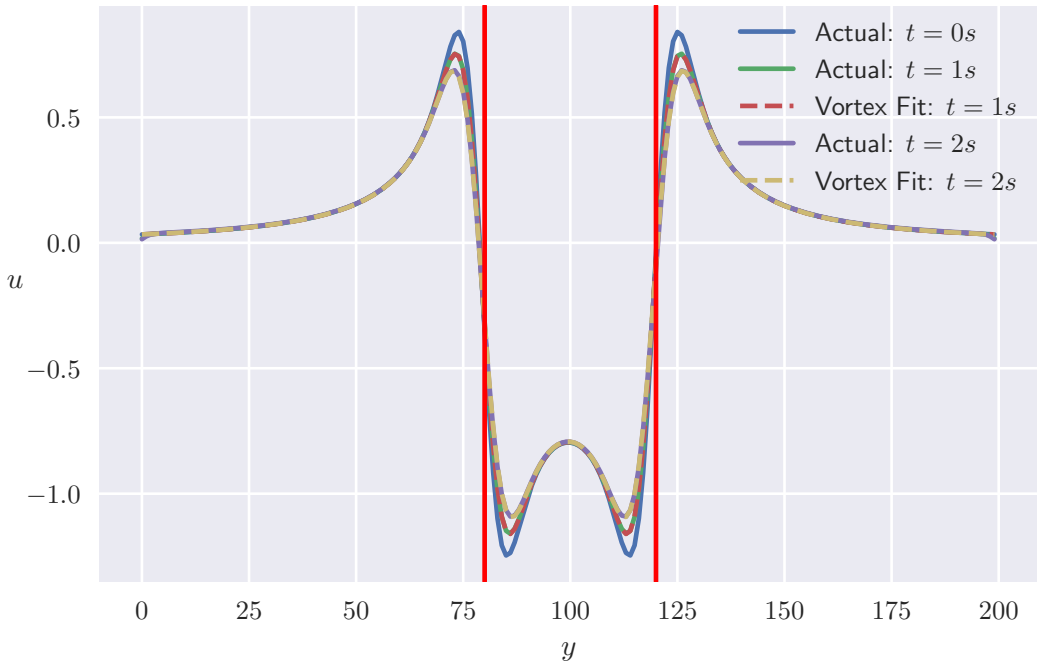


Figure 3.15: Comparison of actual velocity profiles at different time instants with the velocity curves obtained using Vortex-Fit for a 2 particle case of Fig. 3.14 (Vertical red lines indicate the initial position in y-axis for both particles).

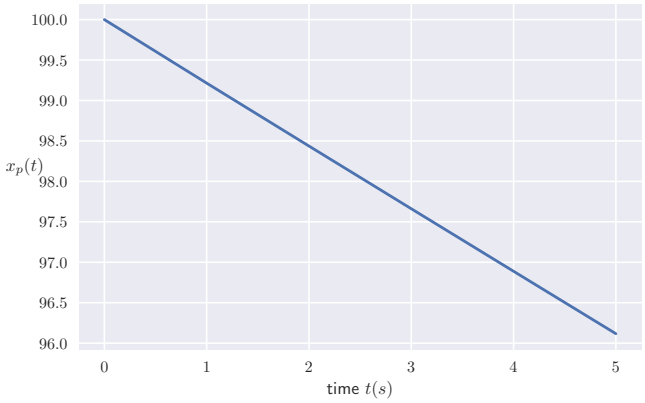
particle. The initial velocity field created by a vortex particle should diffuse with time due to the influence of viscosity of the fluid. It is evident from the actual velocity curves obtained from numerical simulation, as shown in Fig. 3.12. The magnitude of velocity goes on decreasing with time. In terms of vortex particles, it results in a decrease in particle strength and an increase in core size [31], which is exactly the type of behaviour we obtain from our Vortex-Fit operation, as shown in Fig. 3.13.

We also present an example of Vortex-Fit for a case with 2 similar counter-rotating particles, as shown in Fig. 3.14. The two particles are located along the vertical line at $x = 100, y = 80$ and $x = 100, y = 120$ respectively. Both particles have strengths of same magnitude but opposite sign and have same core sizes. Similar to the single particle example in Fig. 3.12, we show the validity of the Vortex-Fit for this 2 particle case in Fig. 3.15. Evolution of both particles will be exactly the same. For this scenario, the velocity induced by one particle on the other is exactly the same and points in the negative direction of x-axis and thus both the particles should be moving together along the negative side of x-axis. Vortex-Fit also exactly reproduces this behaviour of movement for both the particles, as shown in Fig. 3.16.

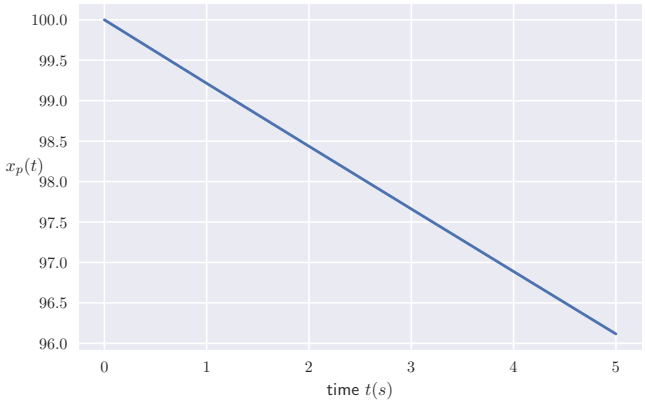
There is a possibility for an argument that Vortex-Fit could be applied throughout the dataset for all the grid based velocity fields delivered by *PhiFlow*. The particle features obtained by Vortex-Fit could then directly used as targets to penalise the neural network predictions during training. However, executing such optimizations for each and every sample in the dataset involves too much of computational effort. On contrary, once we have a trained neural network, Vortex-Fit acts as an excellent analysis tool to directly compare the evolution of particle features predicted by the neural networks for some chosen test cases.

We practically don't lose anything by creating datasets by executing grid-based simulations using *PhiFlow* in order to train neural network for learning Lagrangian vortex dynamics. On the other hand, we take advantage of the simplicity and convenience associated with executing grid based

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) Position of Particle 1: $x_p(t)$.



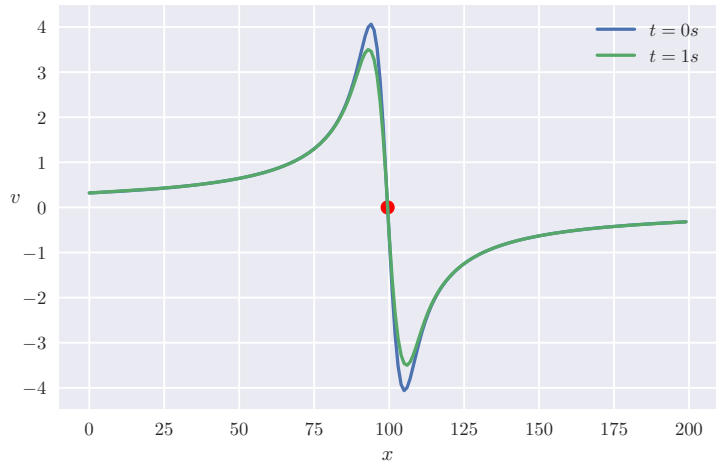
(b) Position of Particle 2: $x_p(t)$.

Figure 3.16: Evolution of 2 similar counter-rotating vortices.

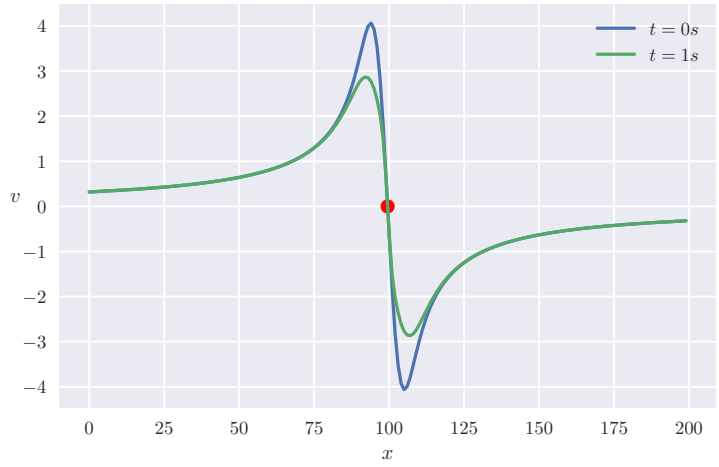
simulations. However, as mentioned earlier, this would be the case only for fluid motion in an open domain with no solid boundaries. We have presented in Section 2.3 of Chapter 2, that given a vorticity field ω, obtaining a divergence-free velocity field $\mathbf{u}$ that satisfies for such a vorticity field, reduces to solving a poisson equation on the velocity field, Eq. 2.14. We have also mentioned earlier that Biot-Savart Law provides a solution in a functional form to such a Poisson equation, which is valid only for open domain scenarios. In presence of solid boundaries, the Poisson equation needs to be solved subjected to a no-through-flow condition at the solid boundaries. There is no functional form of solution to the Poisson equation in such scenarios and thus numerical approaches like Finite Difference Methods or Finite Element Methods are typically employed to solve such equations.

Inviscid flows are subjected to no-through-flow boundary condition and for viscous flows, there exists no-slip boundary condition. Both these conditions are constraints on the velocity field and since the vortex methods uses vorticity ω as the primary variable, there are several difficulties associated with converting these constraints on velocity in terms of vorticity. We have just pointed out the difficulties associated with no-through-flow condition in the previous paragraph. Physically, the no-slip boundary condition expresses the requirement that the flow field must adhere to the boundary. This condition imposes a torque onto the fluid elements adjacent to the wall, which in turn, may impart a rotational motion to the fluid [31]. Hence the no-slip boundary condition is physically manifested by the creation of vorticity at the boundary. Thus, vorticity boundary conditions are often presented as models of vorticity creation rather than rigorous mathematical constraints [13]. Thus, it is evident that it becomes totally non-trivial to

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) $\Delta t = 0.1s$.



(b) $\Delta t = 1.0s$.

Figure 3.17: Demonstration of numerical diffusion in *PhiFlow* for evolution of velocity field created by 1 vortex particle: Velocity profiles at $t = 2s$ for 2 different time step sizes of $\Delta t = 0.1s$ and $\Delta t = 1.0s$.

create datasets and train neural networks that learns to predict vortex particle dynamics and also respect the velocity constraints at the boundaries. However, in our work we take a different approach to solve this issues, which we would present later in this chapter.

3.3 Single Particle Dynamics

A single vortex particle of strength Γ_1, at some specified location $\mathbf{x}_1$, and which has a core size σ_1, produces a velocity field around it which intends to rotate the fluid mass about the the particle location, Fig. 3.3. The vorticity field created by the particle is then obtained by setting $N_p = 1$ in Eq. 3.5, and is written as

$$\omega(\mathbf{x}, t) = \frac{\Gamma_1(t)}{\pi\sigma_1^2(t)} \exp\left(-\frac{|\mathbf{x} - \mathbf{x_1(t)}|^2}{\sigma_1^2(t)}\right), \tag{3.16}$$

https://doi.org/10.24355/dbbs.084-202103121124-0

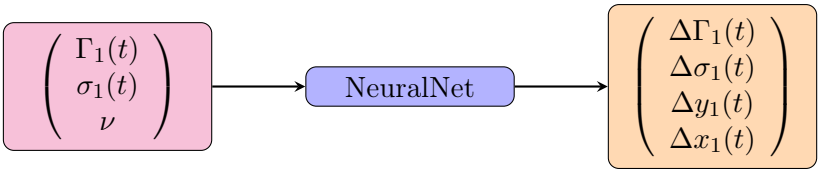


Figure 3.18: Input and output for neural network predicting for single particle dynamics.

and the corresponding velocity field is given by

$$\mathbf{u}(\mathbf{x},t) = \frac{\Gamma_1(t)}{2\pi|\mathbf{x}-\mathbf{x}_1(t)|^2}\left[1-\exp\left(-\frac{|\mathbf{x}-\mathbf{x_1}(\mathbf{t})|^{\mathbf{2}}}{\sigma_1^2(t)}\right)\right]\begin{pmatrix} -(y-y_1(t)) \\ x-x_1(t) \end{pmatrix}. \tag{3.17}$$

Motion of the particles is based on the local velocity field and we have already presented the differential equations for such particle motions in Eq. 2.27. for flows with only one vortex particle, Eq. 2.27 takes the form

$$\frac{d\mathbf{x}_1}{dt} = \mathbf{u}(\mathbf{x}_1(t),t), \quad s.t. \quad \mathbf{u}(\mathbf{x}_1(t_0),t_0) = \mathbf{0}. \tag{3.18}$$

At initial time t_0, velocity at the location of particle is zero, and thus $\mathbf{x}_1(t) = \mathbf{x}_1(t_0)$ is the solution to Eq. 3.18, i.e, the particle has no movement and remains at the same location as its initial location for all times.

For inviscud flows, the material derivative of voriticity is zero $D\omega/Dt = 0$. We have already discussed in previous chapter, that invsicid flows are circulation preserving, and thus the vortex strength of the particle does not change with time and this also to a constant core size. Thus, for single vortex particle in the absence of any viscous effects, the particle location, strength and core size remains the same at all times. However, under the influence of viscosity the flow is diffusive and thus the particle strength goes on decreasing and the core size increases with time.

Grid-based numerical solvers typically suffers from the numerical diffusion. Numerical approximation of convective gradients in the Navier-Stokes equations introduces errors of type which act as an artificial viscosity in such numerical simulation. The effect of numerical diffusion could be reduced to lesser extents by using finer grid resolutions or using lower time step sizes [9]. We demonstrate the extent of numerical diffusion on the data samples generated using *PhiFlow* in Fig. 3.17, where we advance the velocity field generated by single vortex particle using two different time step sizes of $\Delta t = 0.1s$ and $\Delta t = 1.0s$. Ideally the velocity profiles at $t = 2s$ in both Fig. 3.17a and Fig. 3.17b should be exactly the same as the one at $t = 0s$. Due to the presence of numerical diffusion, the velocity field goes on diffusing with time and the extent of it is much higher for a time step of $\Delta t = 1s$ than that for $\Delta t = 0.1s$, as is evident from Fig. 3.17a and Fig. 3.17b respectively. Thus, in general for any flows, with or without viscosity, and for any number of particles, we always consider the particle strengths and core sizes as a dynamic quantity.

We have to decide on the inputs and outputs to a neural network to predict the dynamics of single vortex particle. We use a fully connected network with N_l hidden layers each layer having n_h number of units/neurons. We denote the input vector to neural network as $\mathbf{a}$ and the output from neural network as $\mathbf{b}$. We provide only the particle strength and core size at time t as input to the network. We do not provide the actual location, since for an open domain the evolution of strength and core size of the particle is invariant to the location of the particle. We thus make our neural network predictions for single particle dynamics **location-invariant** and thus inherently respecting the true physics. For viscous flows, it is trivial that we add just one additional parameter corresponding to the kinematic viscosity ν to the input vector $\mathbf{a}$, and we represent this input vector as $\mathbf{a}^{\nu}$. This would also be the case in future when we build input

https://doi.org/10.24355/dbbs.084-202103121124-0

vectors to the neural network for flows with multiple particles. The input vectors always will have one additional element for viscosity in case of viscous flows.

We make the neural network output the change in strength $\Delta\Gamma_1(t)$ and change in core size $\Delta\sigma_1(t)$ from time t to $t + \Delta t$, Fig. 3.18. Even though we know that the particle does not move from its initial location, we still make the network predict the change in positions $\Delta y_1(t)$ and $\Delta x_1(t)$ of the particle. We expect our network to predict a value of zero or close to zero for $\Delta y_1(t)$ and $\Delta x_1(t)$. We show the validity of such predictions in the next chapter. We could therefore write the input and output to neural network as

$$\mathbf{a} = \begin{pmatrix} \Gamma_1(t) \\ \sigma_1(t) \end{pmatrix}, \quad \mathbf{a}^{\nu} = \begin{pmatrix} \Gamma_1(t) \\ \sigma_1(t) \\ \nu \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} \Delta\Gamma_1(t) \\ \Delta\sigma_1(t) \\ \Delta y_1(t) \\ \Delta x_1(t) \end{pmatrix}. \tag{3.19}$$

Then the particle features at next time instant is obtained as

$$\Gamma_1(t + \Delta t) = \Gamma_1(t) + \Delta\Gamma_1(t) \tag{3.20a}$$
$$\sigma_1(t + \Delta t) = \sigma_1(t) + \Delta\sigma_1(t) \tag{3.20b}$$
$$y_1(t + \Delta t) = y_1(t) + \Delta y_1(t) \tag{3.20c}$$
$$x_1(t + \Delta t) = x_1(t) + \Delta x_1(t). \tag{3.20d}$$

3.4 Dynamics of many interacting Vortex Particles

For the dynamics of single vortex particle, we saw that the particle is stationary and the evolution of its strength and core size is solely dependent on its initial counterparts. In case of the fluid state represented by many vortex particles, the evolution of the position, strength and core size of any particle is now not only dependent on its own features but also on the features of all the other particles in the domain. The dynamics of flow is governed by interaction between every pair of particles. We have already presented the coupled system of ordinary differential equations governing the motion of these interacting particles in Eq. 2.27.

A particular particle p in an N_P particle system interacts with all of the remaining $N_p - 1$ particles. This leads to a system where the dynamics of every particle is linked with every other particle in the system. This is very much identical to the classical n-body problems in physics. It is essential that the deep learning models, like the classical physics-based models, perform object-centric and relation-centric reasoning of this system of particles and therefore predict their dynamic behaviours. A system of vortex particles represents an unordered set of data points. As we mentioned earlier, Convolutional Neural Networks are extensively used if the data is available in the form a structured grid, which is the case with grid based velocity fields. However, having an unstructured set of vortex particles requires different strategies to be processed by a neural network.

An unordered system of particles resembles very close to the point clouds data in the fields of computer vision and autonomous driving. These point clouds too represent an unordered set of points in a three dimensional space. Many of the works in computer vision deal with formulating methodologies to still apply the structured convolution operation on the unstructured point clouds [32, 49]. On the other hand, there are several works recently which proposes the application of Graph Neural Networks [4] on the unstructured data. A graph is defined, minimally, as a set of nodes as well as a set of edges adjacent to pairs of nodes. For a system of unordered mesh of objects interacting with one another, nodes represent the actual objects and their associated features, whereas the edges in the graph provides the extent of relationships and interactions between the nodes [5, 44]. Graph Neural Networks implement strong relational inductive biases for learning functions that operate on graphs. Graph Neural Networks, in general do not demand

https://doi.org/10.24355/dbbs.084-202103121124-0

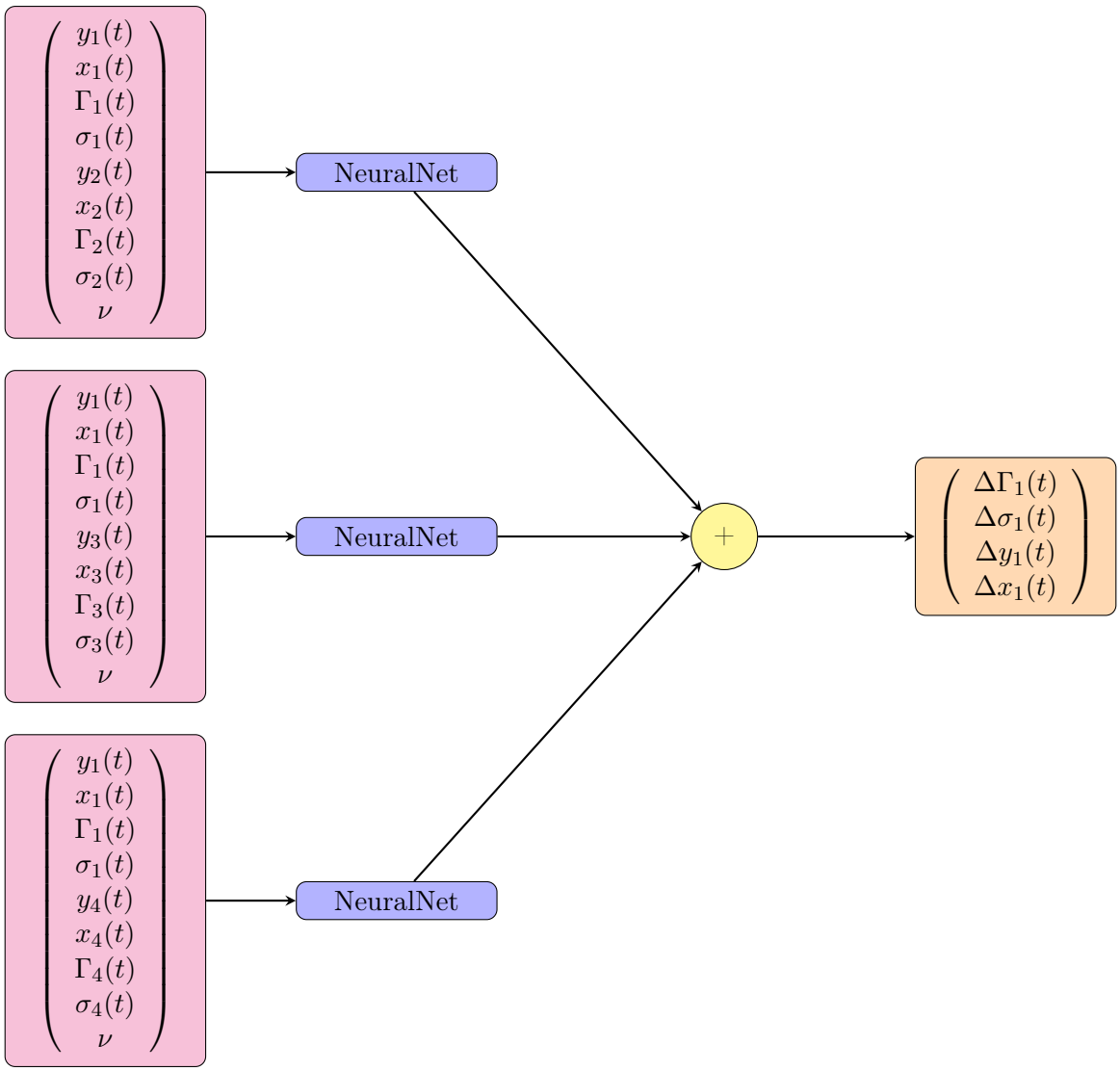


Figure 3.19: An illustration of the application of *Interaction network* for predicting the dynamics of 1^{st} particle in a 4-particle system.

an edge to be always present between two nodes. In fact, there is no need for an edge if there is no interaction between two objects (like data in the form of a tree). However, in an n-particle system, where every particle interacts with every other particle, there is an edge connecting every possible pairs of nodes. Such type of graph neural are referred to as *Interaction networks* [5]. The recent work of Xiong [50] is the closest to our work in terms of the objective of learning the dynamics of vortex particles using deep learning. In order to model and reason about interaction between the vortex particles, they make use of *Interaction networks*. This is the point where our work differs from the work of Xiong [50] in terms of modeling particle interactions. We refer to our networks in general as *Vortex-Networks*. We first present a schematic of a typical *Interaction network* in order to compare with our *Vortex-Network*.

A typical *Interaction network* [5] is comprised of a fully connected network, which takes features vectors of 2 particles concatenated as its input. Thus, in order to say predict the dynamics of particle p, the fully connected network needs to be executed $N_p - 1$ times by providing the concatenated feature vector of particle p with each of the remaining $N_p - 1$ particles. Since the final dynamics of particle p will be based on interaction with each of the remaining particles, the neural network outputs from each of the $N_p - 1$ runs are added together, which acts a final prediction. The same procedure is repeated to predict the dynamics of all the N_p particles.

https://doi.org/10.24355/dbbs.084-202103121124-0

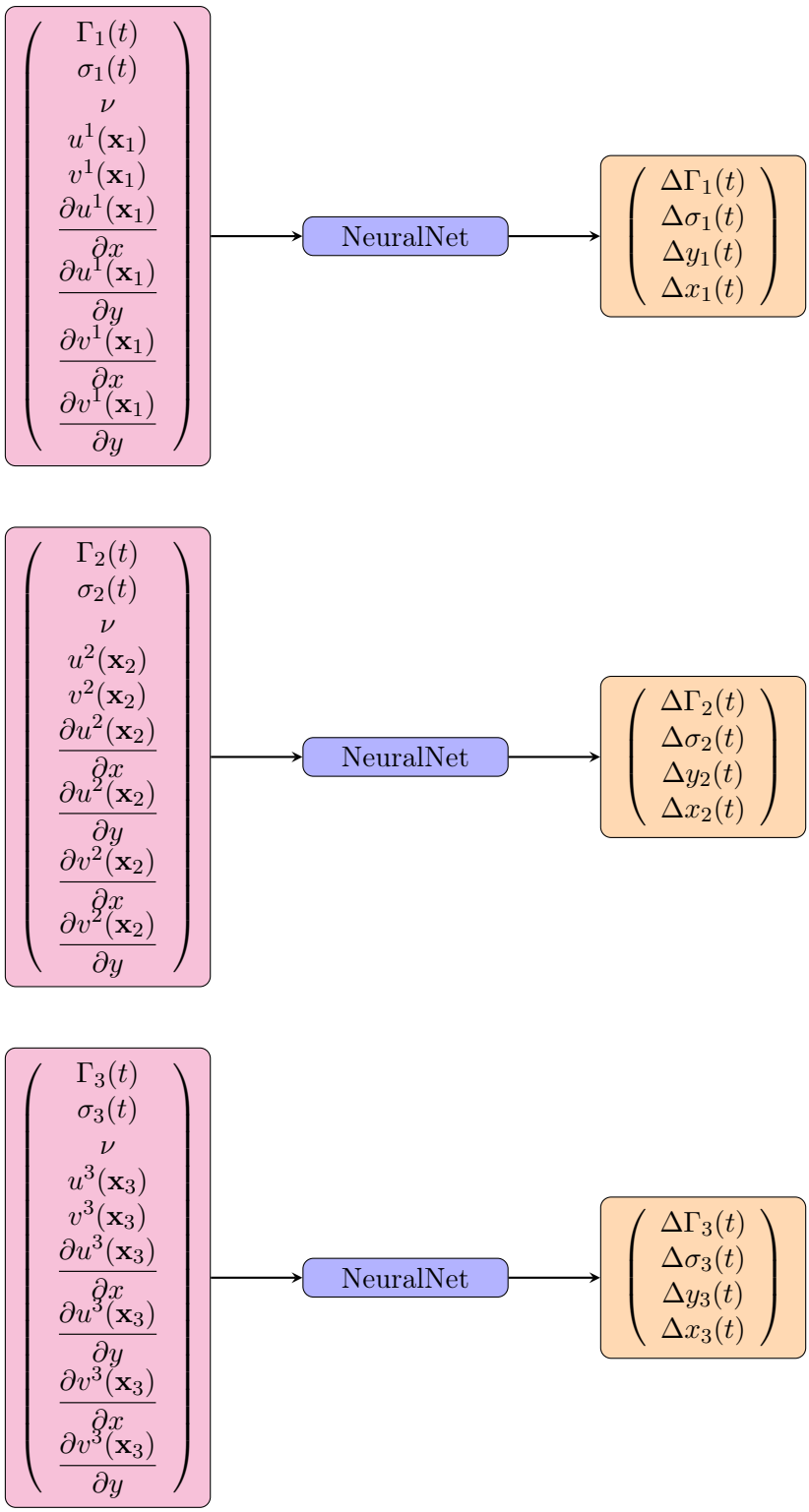


Figure 3.20: An illustration of the application of our *Vortex-Network* using the first order influence vectors for predicting the dynamics of particles in a 3-particle system.

This, it is easy to perceive that such networks model all the possible interactions to predict the final dynamics, but requires $O(N_p^2)$ evaluations of the fully connected network. Fig. 3.19 depicts the modeling of interaction relationships between particles by an *Interaction network*. We only show in Fig. 3.19 for the interaction of a particular particle $p = 1$ with the remaining 3 particles for a system with 4 vortex particles. The influences of each of the 3 remaining particles is modelled by adding up the neural outputs for all of the 3 possible inputs. We then

https://doi.org/10.24355/dbbs.084-202103121124-0

obtain the final estimates of change in the location, strength and core size of particle $p = 1$, which are added up to their values at time t to obtain their counterparts at $t + \Delta t$, as given by Eq. 3.20. In an exact same manner, interaction network is applied to get the corresponding change in particle features $\{\Delta\Gamma_2(t), \Delta\sigma_2(t), \Delta y_2(t), \Delta x_2(t)\}$, $\{\Delta\Gamma_3(t), \Delta\sigma_3(t), \Delta y_3(t), \Delta x_3(t)\}$ and $\{\Delta\Gamma_4(t), \Delta\sigma_4(t), \Delta y_4(t), \Delta x_4(t)\}$ for particles $p = 2$, $p = 3$ and $p = 4$ respectively.

We propose our **Vortex-Network** in comparison to the *Interaction network* to model the particle interactions. In order to predict the dynamics of any particular particle p, we do not need to make $N_p - 1$ neural network runs to obtain the overall contribution from the remaining particles, like in Fig. 3.19. A simplistic thought would suggest that, what about just executing the neural network once to predict the dynamics of particle p? In that case, the input vector to neural network would contain the elements of the feature vector of particle p, but there has to be additional elements which provides a representation of the flow field created by the remaining $N_p - 1$ particles. How do we compute these representations? For a particle p in consideration, let us represent the velocity field created by the other $N_p - 1$ particles as $\mathbf{u}^p$ and we refer to it as the *influence field* for particle p and is written as

$$\mathbf{u}^p(\mathbf{x}) = \sum_{\substack{q=1 \\ q\neq p}}^{N_p} \frac{\Gamma_q}{2\pi|\mathbf{x}-\mathbf{x}_q|^2}\left[1 - \exp\left(-\frac{|\mathbf{x}-\mathbf{x_q}|^2}{\sigma_q^2}\right)\right]\begin{pmatrix} -(y-y_q) \\ x - x_q \end{pmatrix}. \tag{3.21}$$

In order to construct a representation of the influence field $\mathbf{u}^p$, we consider its Taylor series expansion about the location $\mathbf{x}_p$ of particle p. Such an expansion results in

$$\begin{aligned} \mathbf{u}^p_{approx}(\mathbf{x}) = \quad & \mathbf{u}^p(\mathbf{x}_p) + \\ & (x-x_p)\frac{\partial \mathbf{u}^p(\mathbf{x}_p)}{\partial x} + (y-y_p)\frac{\partial \mathbf{u}^p(\mathbf{x}_p)}{\partial y} + \\ & \frac{1}{2}\left[(x-x_p)^2\frac{\partial^2 \mathbf{u}^p(\mathbf{x}_p)}{\partial x^2} + 2(x-x_p)(y-y_p)\frac{\partial^2 \mathbf{u}^p(\mathbf{x}_p)}{\partial x \partial y} + (y-y_p)^2\frac{\partial^2 \mathbf{u}^p(\mathbf{x}_p)}{\partial y^2}\right] + \\ & O\big((x-x_p)^3\big) + O\big((y-y_p)^3\big). \end{aligned} \tag{3.22}$$

We construct our influence vector $\mathbf{i}^p$ for particle p using the coefficient terms in the Taylor series expansion, Eq. 3.22 corresponding to different orders of spatial derivatives of the influence field $\mathbf{u}^p$ evaluated at the particle location $\mathbf{x}_p$. Ideally, if we have the coefficients corresponding to all the infinite terms in the expansion, we could provide an exact representation of the influence field $\mathbf{u}^p$. However, practically this would be impossible and therefore we consider only the limited set of coefficients. We construct the influence vectors of different orders,

https://doi.org/10.24355/dbbs.084-202103121124-0

$$\mathbf{i}_0^p = \begin{pmatrix} u^p(\mathbf{x}_p) \\ v^p(\mathbf{x}_p) \end{pmatrix}, \quad \mathbf{i}_1^p = \begin{pmatrix} u^p(\mathbf{x}_p) \\ v^p(\mathbf{x}_p) \\ \frac{\partial u^p(\mathbf{x}_p)}{\partial x} \\ \frac{\partial u^p(\mathbf{x}_p)}{\partial y} \\ \frac{\partial v^p(\mathbf{x}_p)}{\partial x} \\ \frac{\partial v^p(\mathbf{x}_p)}{\partial y} \end{pmatrix}, \quad \mathbf{i}_2^p = \begin{pmatrix} u^p(\mathbf{x}_p) \\ v^p(\mathbf{x}_p) \\ \frac{\partial u^p(\mathbf{x}_p)}{\partial x} \\ \frac{\partial u^p(\mathbf{x}_p)}{\partial y} \\ \frac{\partial v^p(\mathbf{x}_p)}{\partial x} \\ \frac{\partial v^p(\mathbf{x}_p)}{\partial y} \\ \frac{\partial^2 u^p(\mathbf{x}_p)}{\partial x^2} \\ \frac{\partial^2 u^p(\mathbf{x}_p)}{\partial x \partial y} \\ \frac{\partial^2 u^p(\mathbf{x}_p)}{\partial y^2} \\ \frac{\partial^2 v^p(\mathbf{x}_p)}{\partial x^2} \\ \frac{\partial^2 v^p(\mathbf{x}_p)}{\partial y \partial x} \\ \frac{\partial^2 v^p(\mathbf{x}_p)}{\partial y^2} \end{pmatrix}, \tag{3.23}$$

where the subscripts 0, 1 and 2 in the influence vectors $\mathbf{i}_0^p$, $\mathbf{i}_1^p$ and $\mathbf{i}_2^p$ indicate that the influence vector is made up of the coefficient terms upto the zeroth, first and second order derivatives of the influence field respectively. However, one could practically construct influence vectors of any order. Increasing the order of the influence vectors mathematically provides better and better approximation of the actual influence field.

Our input vector to the neural network is obtained by concatenating the feature vector of particle p with the elements of the influence vector. This vector now has the information about the particle itself as well as the influence of all the remaining particles. Our neural network makes use of such an input vector to make predictions for the change in particle features over the next time step. In an exact same way, we predict the dynamics of all the particles by constructing their respective influence vectors. Fig. 3.20 demonstrates the application of our *Vortex-Network* to predict the evolution dynamics of all the particles in a 3-particle system. We could notice from Fig. 3.20 that the actual particle location $\mathbf{x}_p$ is not included in the input vector to the neural network. This is due to the fact that, given the particle strength Γ_p and core size σ_p and the influence vector $\mathbf{i}^p$, the motion of the particle and the evolution of its strength and core size should be invariant to the actual particle location for an open domain scenario, and therefore we again make our networks **location-invariant**. By providing the neural network with our influence vector, we force our neural networks to make predictions based on much more physics based information, rather than just providing raw pairs of particle features, which was the case with *Interaction networks*. Also, in contrast to the *Interaction networks*, we only need $O(N_p)$ executions of the fully connected network to obtain the prediction dynamics over a time step. Thus, the computational effort required is much lower, the effect of which becomes more and more significant with increasing number of vortex particles.

In order train the *Vortex-Network*, we penalise the error on the grid-based velocity field as a result of the vortex particle representation predicted by the *Vortex-Network*. We use the squared $L2$ norm of the difference between the predicted and true grid-based velocity field as the loss

https://doi.org/10.24355/dbbs.084-202103121124-0

function for training. For training with batch size N_b, the loss function is given by

$$L_{vortex} = \frac{1}{N_b} \sum_{i=1}^{N_b} \|\mathbf{U^G}_i(t) - \widetilde{\mathbf{U}}_i^{\mathbf{G}}(t)\|^2, \tag{3.24}$$

where $\widetilde{\mathbf{U}}^{\mathbf{G}}$ and $\mathbf{U}^{\mathbf{G}}$ are the predicted and true grid-based velocity fields respectively.

3.5 Boundary Condition Network

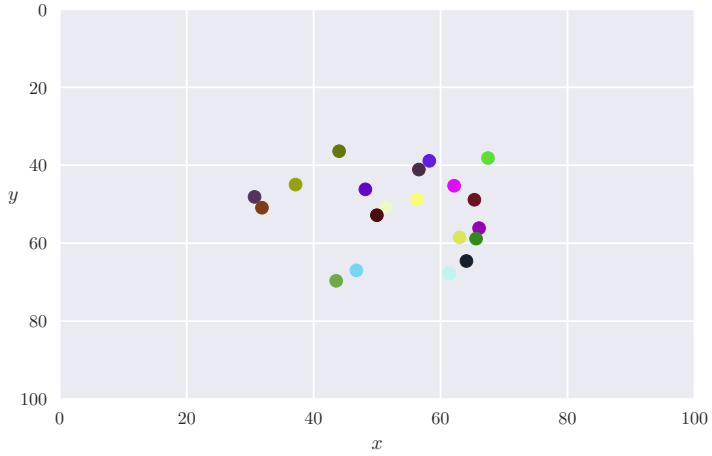
So, far we have only considered flows in an open domain. But, how would the dynamics of the vortex particles alter in presence of solid boundaries? We have already discussed in detail in section 3.2 about the key technical difficulties associated with classical Vortex Particle Methods in regards to satisfying for the appropriate boundary conditions. It therefore becomes naturally difficult to make the neural networks predict the particle dynamics and also respect the appropriate boundary conditions. Since we are using *PhiFlow* to generate data samples, it is possible to generate data samples corresponding to flows in presence of solid boundaries. In cases with open domain, it was actually possible to use the grid based velocity fields from *PhiFlow* to learn for vortex particle dynamics, due to the mappings introduced by using Biot-Savart Law. This is not the case when we have solid boundaries. Thus, it becomes difficult to enforce the constraint pertaining to boundary conditions on neural networks that predicts the vortex particle dynamics.

We already have our *Vortex-Network*, which predicts the particle dynamics for an open domain. We propose an approach based on learning a correction velocity field on top of the velocity field corresponding to the vortex particle representations delivered by our *Vortex-Network*. In our work, we only consider the solid boundaries, that enclose our two dimensional square domain D. We do not consider the presence of any internal boundaries or obstacles. Also, we would consider in this work only the procedures to deal with the no-through-flow boundary condition, and thus in presence of boundaries, we only consider inviscid flows in this work. Our objective here is, given the vortex particle representation which produces a velocity field $\mathbf{u}_{vortex}$ valid under an open domain setting, we try to add a correction field $\mathbf{u}_{corr}$, such that the total field of $\mathbf{u}_{vortex} + \mathbf{u}_{corr} = \mathbf{u}_{total}$ satisfies the inviscid boundary condition at the solid boundaries. In terms of actual flow physics, the difference between $\mathbf{u}_{vortex}$ and $\mathbf{u}_{total}$ is the *pressure solve* operation. We have already mentioned in section 2.2 of previous chapter, that the *pressure solve* step is responsible for making the velocity field divergence free, while satisfying the no-through-flow boundary condition. Fig. 3.21 demonstrates the application of *pressure solve* on the velocity field (Fig. 3.21b and Fig. 3.21d) generated by a particular configuration of vortex particles, Fig. 3.21a. It is evident from Fig. 3.21c, which represents the x-component of velocity field after *pressure solve*, that the vertical boundaries have the x-component of velocity being zero. Similarly, in the horizontal boundaries, the components are zero, as shown in Fig. 3.21e.

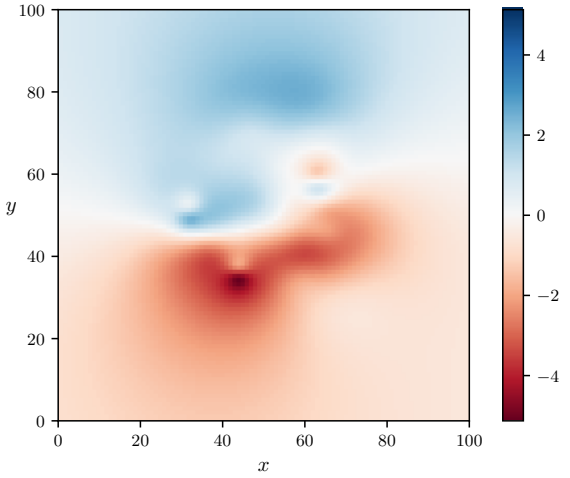
In our case the velocity field $\mathbf{u}_{vortex}$ is already divergence free, since it is produced by sets of vortex particles. If we provide such a $\mathbf{u}_{vortex}$ to *PhiFlow* and apply the *pressure solve* on it, the resulting velocity field $\mathbf{u}_{total}$ too is divergence-free and it obeys the invisicid boundary conditions. Thus, the velocity obtained by taking the difference between $\mathbf{u}_{total}$ and $\mathbf{u}_{vortex}$, which is the correction velocity field $\mathbf{u}_{corr}$ is also divergence-free. Since, we could already predict $\mathbf{u}_{vortex}$ from the vortex particle dynamics predicted using our *Vortex-Network*, we only need to formulate a strategy using neural network that predicts the appropriate correction field $\mathbf{u}_{corr}$. We refer to our network predicting the correction field as **BC-Net**. In order to train our *BC-Net*, we generate dataset $\mathbf{D}_{BC}$ comprising of pairs $\{\mathbf{u}_{vortex}, \mathbf{u}_{total}\}$ of the velocity field by vortex particles and their counterparts obtained after the *pressure solve* step.

Theoretically, the correction velocity $\mathbf{u}_{corr}(\mathbf{x})$ at any point $\mathbf{x}$ depends on the position of that point with respect to the boundaries and the velocity field due to vortex particles $\mathbf{u}_{vortex}$ as a

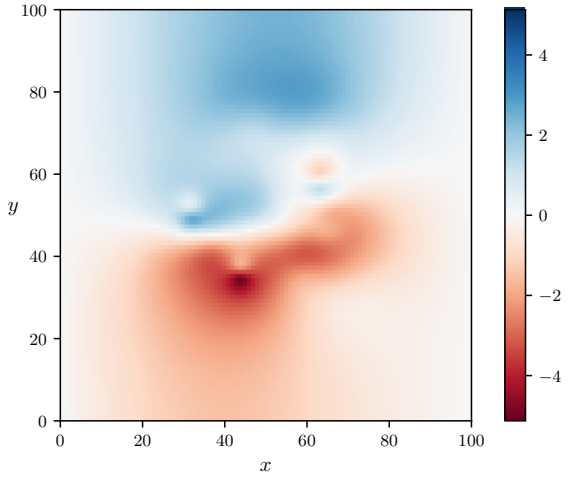
https://doi.org/10.24355/dbbs.084-202103121124-0



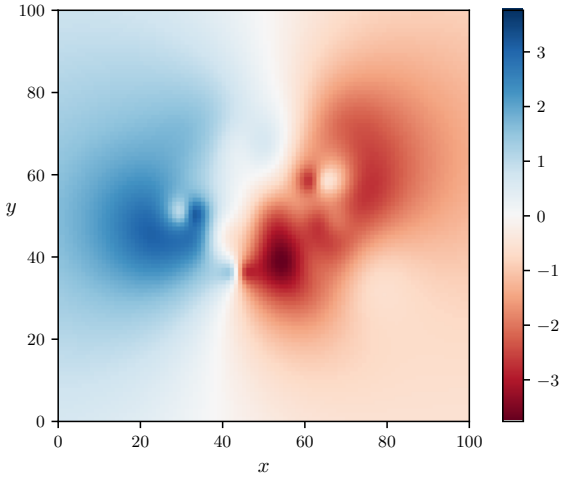
(a) Particle Locations.



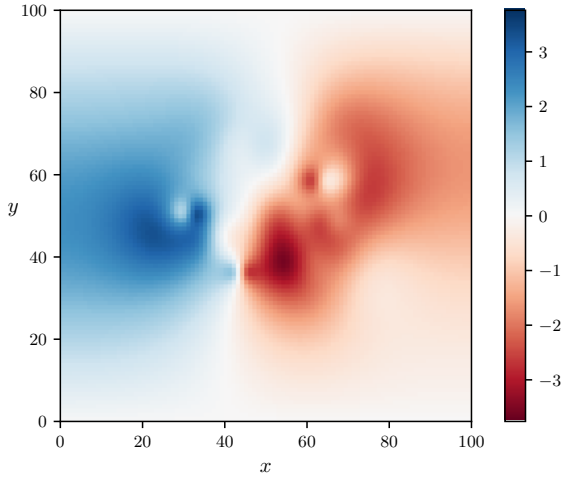
(b) Velocity field by vortex particles (x-component): no boundaries.



(c) Velocity field after *pressure solve* (x-component): with boundaries.



(d) Velocity field by vortex particles (y-component): no boundaries.



(e) Velocity field after *pressure solve* (y-component): with boundaries.

Figure 3.21: Comparison of velocity field produced by set of vortex particles in an open domain with the velocity field obtained after application of *pressure solve* operation on it

https://doi.org/10.24355/dbbs.084-202103121124-0

whole in general. So, there is a need to encode the representation of the velocity field $\mathbf{u}_{vortex}$ at point $\mathbf{x}$. We draw motivation from the process of constructing influence vectors for our *Vortex-Network*. We provide the network with different orders of spatial derivatives of $\mathbf{u}_{vortex}$ evaluated at the point $\mathbf{x}$. In addition, we also present the actual location $\mathbf{x}$ as input to the neural network. Since we consider a domain of fixed length L, providing the network with the actual co-ordinate of the point $\mathbf{x}$ provides a representation of its vicinity at the boundary. Thus we have following input vectors of different order to our *BC-Net*,

$$\mathbf{i}_0^{BC} = \begin{pmatrix} x \\ y \\ u_{vortex}(\mathbf{x}) \\ v_{vortex}(\mathbf{x}) \end{pmatrix}, \quad \mathbf{i}_1^{BC} = \begin{pmatrix} x \\ y \\ u_{vortex}(\mathbf{x}) \\ v_{vortex}(\mathbf{x}) \\ \frac{\partial u_{vortex}(\mathbf{x})}{\partial x} \\ \frac{\partial u_{vortex}(\mathbf{x})}{\partial y} \\ \frac{\partial v_{vortex}(\mathbf{x})}{\partial x} \\ \frac{\partial v_{vortex}(\mathbf{x})}{\partial y} \end{pmatrix}, \quad \mathbf{i}_2^{BC} = \begin{pmatrix} x \\ y \\ u_{vortex}(\mathbf{x}) \\ v_{vortex}(\mathbf{x}) \\ \frac{\partial u_{vortex}(\mathbf{x})}{\partial x} \\ \frac{\partial u_{vortex}(\mathbf{x})}{\partial y} \\ \frac{\partial v_{vortex}(\mathbf{x})}{\partial x} \\ \frac{\partial v_{vortex}(\mathbf{x})}{\partial y} \\ \frac{\partial^2 u_{vortex}(\mathbf{x})}{\partial x^2} \\ \frac{\partial^2 u_{vortex}(\mathbf{x})}{\partial x \partial y} \\ \frac{\partial^2 u_{vortex}(\mathbf{x})}{\partial y^2} \\ \frac{\partial^2 v_{vortex}(\mathbf{x})}{\partial x^2} \\ \frac{\partial^2 v_{vortex}(\mathbf{x})}{\partial y \partial x} \\ \frac{\partial^2 v_{vortex}(\mathbf{x})}{\partial y^2} \end{pmatrix}. \tag{3.25}$$

BC-Net then outputs just the x-component and y-component of the correction velocity corresponding to point $\mathbf{x}$ in the input vector. We refer to this predicted correction velocity as $\widetilde{\mathbf{u}}_{corr}(\mathbf{x})$. We need to penalize this prediction with the target $\mathbf{u}_{corr}(\mathbf{x})$ for training the *BC-Net*. We have the velocity fields $\mathbf{u}_{vortex}$ and $\mathbf{u}_{total}$ from our dataset, which we obtain from *PhiFlow*, and thus is available in a grid based format. Thus, we cannot have the target correction velocity for any point $\mathbf{x}$ in the domain, rather we have it only for the points corresponding to the centers of grid cells, which we refer to as $\mathbf{x}^g$. Say we sample N^g grid cell points, then we compute the mean squared error loss on the correction velocity predictions as follows

$$L_{mse} = \frac{1}{N^g} \sum_{i=1}^{N^g} \| \mathbf{u}_{vortex}(\mathbf{x}_i^g) + \widetilde{\mathbf{u}}_{corr}(\mathbf{x}_i^g) - \mathbf{u}_{total}(\mathbf{x}_i^g) \|^2. \tag{3.26}$$

It is necessary that the the correction field predicted by the neural network should be divergence free. We take inspiration from the work of Raiss [40] on *Physics-Informed Neural Networks* (PINN). A PINN is simply a neural network, the training of which is based not only on the mean squared error penalisation of the neural networks, but also on additional loss terms which makes the network predictions obey some fundamental physical laws concerning the problem in hand. Such physical constraints functions as a prior information to the training algorithm and enforces strong inductive biases in the learned parameters of the neural network [39]. The physical constraints act as a regularization agent that constrains the space of admissible solutions to a manageable size (for e.g., for our *BC-Net* we need to discard any non realistic flow solutions

https://doi.org/10.24355/dbbs.084-202103121124-0

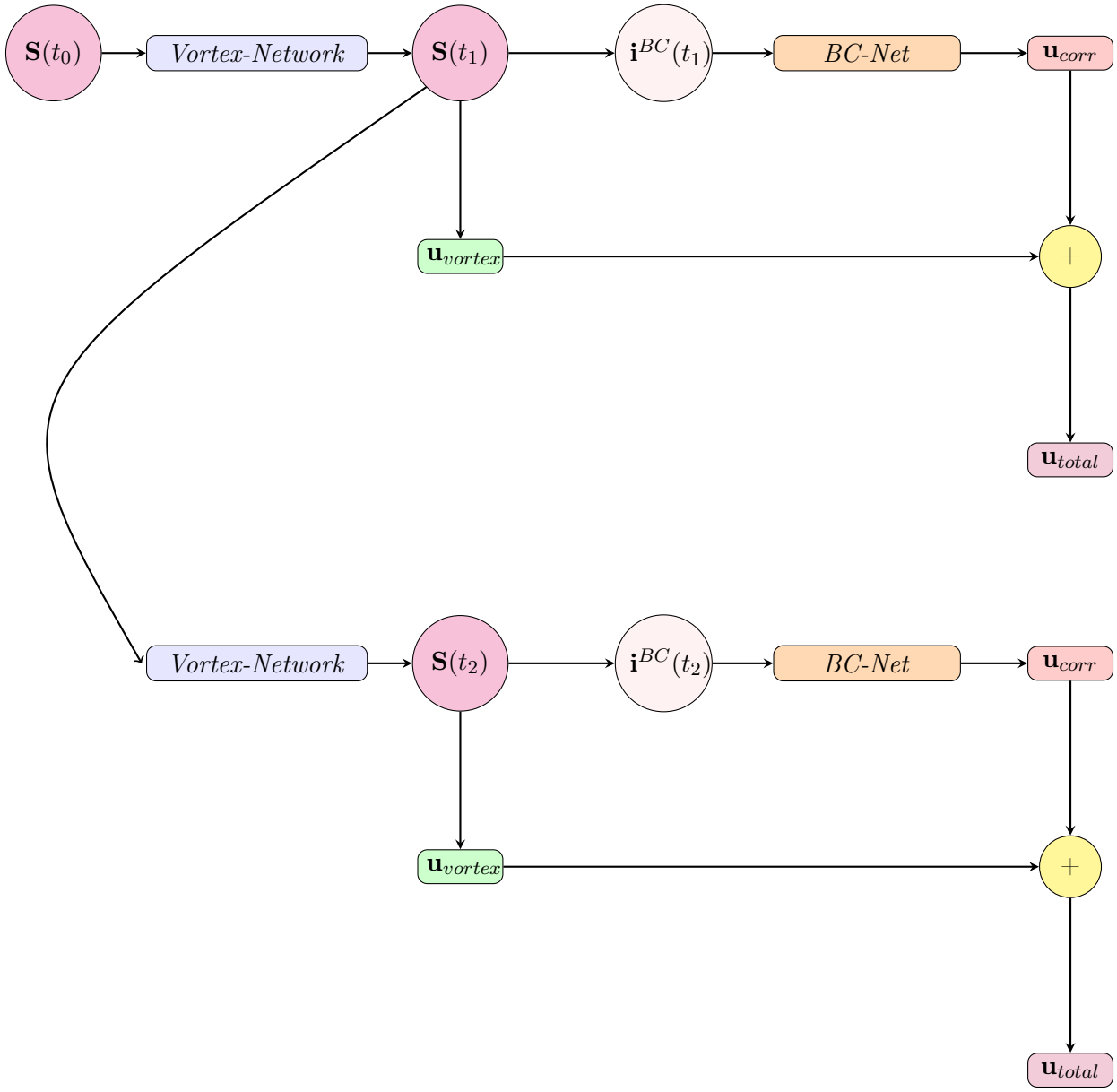


Figure 3.22: An illustration of *Vortex-Network* and *BC-net* in action together for predicting flow dynamics in presence of boundaries.

that violate the divergence free constraint) [40]. In return, encoding such structured information into a learning algorithm results in amplifying the information content of the data that the algorithm sees, enabling it to quickly steer itself towards the right solution and generalize well even when only a few training examples are available [38].

We penalise our *BC-Net* predictions to be divergence-free. There is now no restriction on the point $\mathbf{x}$ for which we could compute this loss. We could penalise for both grid and non-grid points. During training, we sample N^{ng} non-grid points in addition to N^g grid points that we have already sampled and apply the divergence-loss on both these sets of points, as below

$$L_{div} = \frac{1}{N^g} \sum_{i=1}^{N^g} \|\nabla \cdot \widetilde{\mathbf{u}}_{corr}(\mathbf{x}^g)\|^2 + \frac{1}{N^{ng}} \sum_{i=1}^{N^{ng}} \|\nabla \cdot \widetilde{\mathbf{u}}_{corr}(\mathbf{x}^{ng})\|^2. \tag{3.27}$$

Finally, the correction field predictions when summed up with the velocity field induced by vortex particles $\mathbf{u}_{vortex}$ should respect the no-through-flow boundary conditions. Therefore, we compute

https://doi.org/10.24355/dbbs.084-202103121124-0

another loss term, which we refer to as boundary-condition loss. We sample N^b points located at the solid boundaries of the domain, and apply the boundary-condition loss as follows

$$L_{boundary} = \frac{1}{N^b} \sum_{i=1}^{N^g} \| \big(\mathbf{u}_{vortex}(\mathbf{x}_i^b) + \widetilde{\mathbf{u}}_{corr}(\mathbf{x}_i^b) \big) \cdot \mathbf{n} \|, \tag{3.28}$$

where $\mathbf{n}$ is the unit normal to the boundary. Thus, the final loss function for training *BC-Net* is just a summation of all the three loss terms above presented, whcih is given by

$$L_{bc} = L_{mse} + L_{div} + L_{boundary}. \tag{3.29}$$

We have our *Vortex-Network* already trained to predict the vortex particle dynamics in an open domain. Now, we also have our *BC-net* which predicts appropriate correction velocities to satisfy for the flow dynamics in presence of boundaries. Only during the evaluation/prediction phase we combine both the networks in order to make a combined prediction of particle dynamics and the corresponding correction fields, as shown in Fig. 3.22. We first state from the vortex particle representation $\mathbf{S}(t_0)$ at time t_0. We then compute the influence vectors for each of the particles to construct the input vector for the *Vortex-Network*. Using *Vortex-Network* predictions, we then have the corresponding representation at $\mathbf{S}(t_1)$ at time t_1. We use this representation to construct the input vector $\mathbf{i}^{BC}$ to our *BC-Net* and predict the corresponding corresponding correction field. For predicting the fluid state at next time step t_2, we follow exactly the same procedure starting from the vortex particle representation $\mathbf{S}(t_1)$.

We have presented and discussed in detail in this Chapter, the strategies for dataset generation, modeling particle interactions and dealing with appropriate boundary conditions, in the context of using neural networks to learn the dynamic behaviour of Lagrangian vortex particles. In the next Chapter, we will present the results of the experimentations that we performed in order to test our proposed approaches.

https://doi.org/10.24355/dbbs.084-202103121124-0

Chapter 4

Results and Discussions

In this Chapter, we will present the experimental results of our proposed approaches for learning Lagrangian vortex particle dynamics using neural networks. For all the experiments, we consider a square domain consisting of 120120 cells of unit length. We use a time step of $\Delta t = 0.2s$ for executing all the numerical simulations. We sample the core sizes σ_p for the particles uniformly in a range of 2 to 10 grid cells. In order to sample for the values of particle strengths, we consider the the maximum velocity induced by a particle with core size σ_p. This is easily achievable by considering the expression for the velocity field produced as a result of a single vortex particle Eq. 3.17. The magnitude of maximum velocity is numerically equal to $v_{max} = 0.1016\Gamma_p/\sigma_p$ [6]. In order to avoid large velocity values, we prefer to constraint v_{max} to be between 0.5 and 2, with an objective that for $\Delta t = 1s$, any fluid particle in the domain does not have a displacement $v_{max}\Delta t$ of more than 2 grid cells. For a given core size σ_p, we therefore randomly sample a factor between 0.5 and 2 corresponding to the magnitude of maximum velocity and obtain the strengths Γ_p by substituting this factor in the expression for v_{max}. For viscous flows, the kinematic viscosity are sampled uniformly in the range of 0 to 3.0

For the fully connected network in the *Vortex-Network*, we use 5 hidden layers with each layer having 100 hidden units. We use a Leaky Rectified Linear Unit activation (LeakyReLU) as a non-linear activation function in each of the hidden layers. For normalization, we use the Layer Normalization (LayerNorm) [3] after the activation function in each of the hidden layer, instead of the Batch Normalization (BatchNorm) [25] layer. We could roll out our neural network multiple times in an recurrent manner to train for multiple number of time steps. This is similar in way to the Long-Short-Term-Memory (LSTM) networks, where BatchNorm is typically not used. This is due to the fact that, in a BatchNorm layer, the statistics for normalization in each layer are computed per batch, and this does not consider the recurrent part of the network. Weights are shared in an LSTM, and the activation response for each recurrent loop might have completely different statistical properties. This is the reason we, too, avoid using BatchNorm, even though our network is a simple fully connected network instead of an LSTM.

We create different datasets in order to perform experiments for different flow scenarios. Each of the datasets consists of 4000 data samples, with 60% of the data samples used for training, an another 20% for validation and the remaining 20% for testing. We create the following 3 datasets

- A dataset with 10 vortex particles without any viscous effects to present a demonstration of our approach based on the proposed *Vortex-Network* for modeling particle-particle interactions. We refer to the dataset as **10-Particle-Dataset**. We train several variants of *Vortex-Network* with this dataset corresponding to different orders of the influence vector $\mathbf{i}^p$. We train for the orders of 0, 1, 2 adn 3, and we refer to these networks as **Vortex-Network-0**, **Vortex-Network-1**, **Vortex-Network-2** and **Vortex-Network-**

https://doi.org/10.24355/dbbs.084-202103121124-0

3 respectively. We also train the **Interaction-Network** using this dataset, in order to compare the results with the *Vortex-Networks*.

- A dataset with 10 vortex particles, but this time with viscosity and we refer to it as **10-Particle-Viscous-Dataset**. We train our proposed *Vortex-Network* with an influnece vector of order 2, and refer to this network as **Vortex-Network-Viscous**.
- A dataset for flows in presence of solid boundaries with pairs of velocity field by 10 vortex particles and its counterpart obtained after the *pressure solve* step, as presented in section 3.5 for training the boundary condition network **BC-Net**. We refer to this dataset as **BC-Dataset**.

All the neural networks are trained using Adam [27]. The networks are trained with a learning rate of $1e$-3 for the first 150 epochs and then reduced by a factor of 10 after every 50 epochs, till we reach a learning rate of $1e$-6. A batch size of 32 is used during training and the $L2$ regularization parameter is set to $1e$-5. We make the networks learn to predict for the change in the states of vortex particles corresponding to a time step of $\Delta t = 1s$.

Even though, we train the *Vortex-Networks*, with a dataset made up of with 10 vortex particles, the network could theoretically be evaluated for any number of particles. Table 4.1 shows the comparison in performances between the **Vortex-Networks** of order 0, 1, 2 and 3 and the *Interaction-Network*. The performance metric used here is the $l2$ norm of the error, Eq. 3.24, between the actual velocity field and the velocity field as a result of the vortex dynamics predictions by our neural networks. A mean squared error (MSE) estimate of the $l2$ norm of the error of individual data samples is computed to estimate the performance over a dataset. We show the metrics in Table 4.1 and in the bar plot in Fig. 4.1 for the predictions over a single time step.

Network	train	val	test
Vortex-Network-0	72.52	76.97	82.34
Vortex-Network-1	59.06	68.08	70.91
Vortex-Network-2	46.63	52.17	55.18
Vortex-Network-3	44.81	51.51	54.52
Interaction-Network	33.59	36.41	37.98

Table 4.1: Tabulated comparison between the variants of *Vortex-Network* and the *Interaction-Network* using the Mean Squared Error (MSE) on the velocity field averaged over all the data samples.

It is evident from Table 4.1 that increasing the order of the influence vectors for the *Vortex-Network* form 0 to 3 decreases the MSE obtained on the velocity field. We expect this behaviour, because the greater the order of the influence vector, the more information a single particle has about the influence field created by the other particles. Thus the neural network learns on top of more and more enriched information content with higher order influence vectors. However, the improvement in performance form order 2 to 3 is very minor. It is also evident that the MSE from the *Interaction-Network* is better than the best *Vortex-Network* of order 3. This is because, in case of *Interaction-Network*, the neural network has the complete information about the influence on a particular particle by all the other particles, whereas in case of our *Vortex-Network*, the networks base their predictions only on the basis of partial information, which gets more and more enriched with increasing orders of influence vectors. From now on for all the visualisations, we will be using the predictions from **Vortex-Network-2**.

In order to make predictions with the *Vortex-Networks* for a number of time steps, we just roll out the network for as many time steps as desired. Fig. 4.2 shows a visual depiction of the comparison between the velocity field from numerical simulation with the one computed from

https://doi.org/10.24355/dbbs.084-202103121124-0

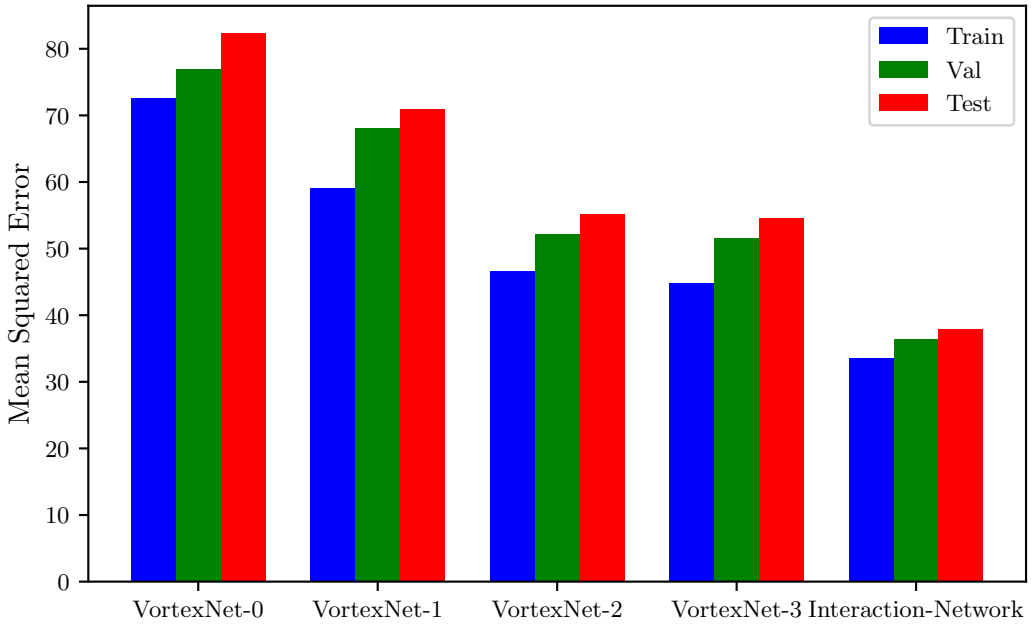


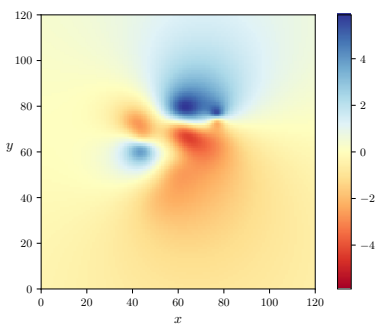
Figure 4.1: Comparison between the variants of *Vortex-Network* and the *Interaction-Network* using the Mean Squared Error (MSE) on the velocity field averaged over all the data samples: Bar Plot.

the vortex particle dynamics predictions from our neural networks. Here the network predicts for change of state form $t = 0s$ to $t = 1s$. From Fig. 4.3, one could notice the differences between the true velocity fields at $t = 0s$ and $t = 1s$. Evidently, the predictions accurately match the true velocity fields at $t = 1s$. In order to predict for $t = 2s$, we just pass the outputs computed for $t = 1$ back into the network, the results of which is shown in Fig. 4.3. Also, one could notice that the maximum value of mean squared error in the error map Fig. 4.3j is higher than that in the error map in Fig. 4.2j. This is due to the fact that errors usually adds up over the time steps when rolling out these networks to make predictions for multiple time steps.

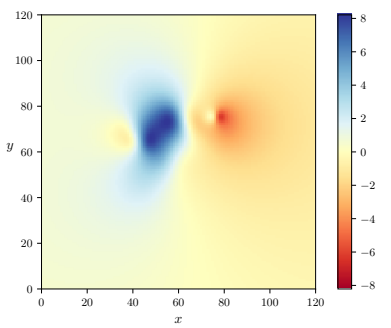
We will now present and compare the evolution predictions using our *Vortex-Network* for a couple of interesting flow scenario with 2 vortex particles. First, we consider 2 particles located along the same vertical line with coordinates $x_1 = 60$, $y_1 = 40$ and $x_2 = 60$, $y_2 = 80$. They both have a same core size of $\sigma_1 = \sigma_2 = 5$. Both the particles have strengths of same magnitude, but of opposite sign $\Gamma_1 = 100$, $\Gamma_2 = -100$. Therefore, we have a case of 2 counter-rotating vortices. Since, they initially, lie along a same vertical line, these particles should move together along the horizontal axis. This is evident from Fig. 4.4, which shows the velocity magnitudes obtained by numerical solution for upto $t = 20s$. One could notice that the two particles move slowly along the negative side of the x-axis. For the given particle configuration, we also obtain the predictions from our *Vortex-Network* for up-to $t = 20s$, the velocity magnitudes of which is shown in Fig. 4.5. One could also visualise this movement along the negative x-axis from Fig. 4.5.

Fig. 4.6 shows the curves for evolution of both the particles, along with the evolution of their strengths and core sizes. We compare these curves with the ones obtained by *Vortex-Fit*. Since the interesting movement here is the horizontal one, it is evident from Fig. 4.6a and Fig. 4.6b that the predicted locations x_p matches very closely to the ones obtained by *Vortex-Fit*, but the deviation increases as the time increases. The plots for vertical movement Fig. 4.6c and Fig. 4.9d

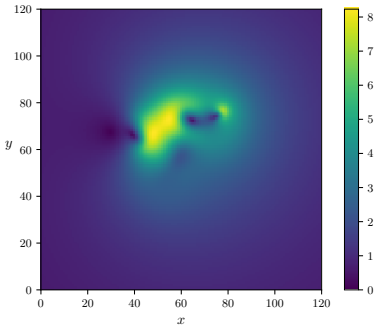
https://doi.org/10.24355/dbbs.084-202103121124-0



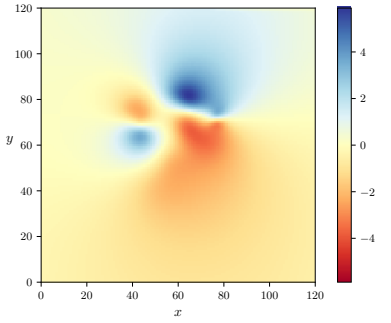
(a) Velocity (x-component) $t = 0s$.



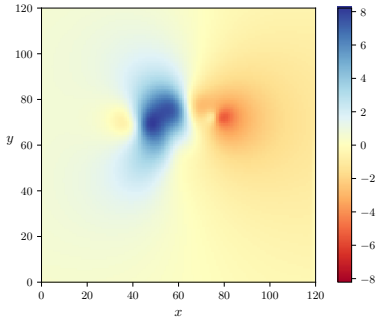
(b) Velocity (y-component) $t = 0s$.



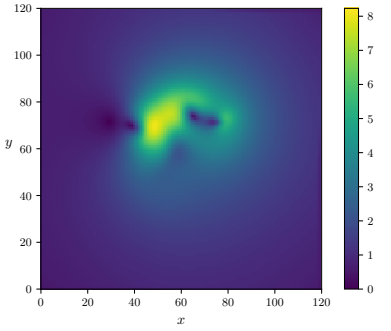
(c) Velocity Magnitude $t = 0s$.



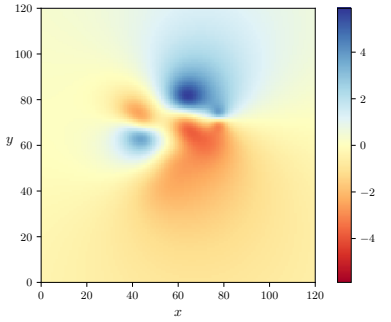
(d) Velocity (x-component): Simulation $t = 1s$.



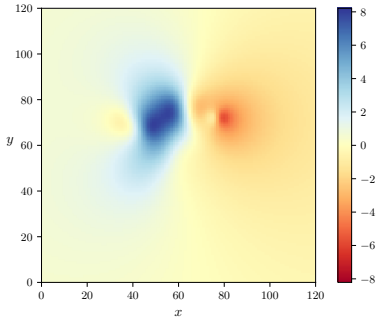
(e) Velocity (y-component): Simulation $t = 1s$.



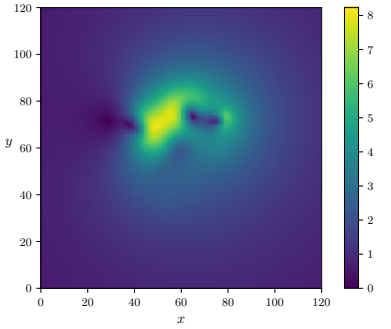
(f) Velocity Magnitude: Simulation $t = 1s$.



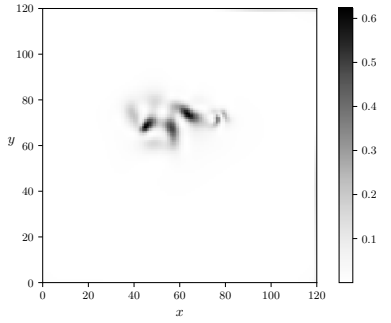
(g) Velocity (x-component): Neural Network $t = 1s$.



(h) Velocity (y-component): Neural Network $t = 1s$.



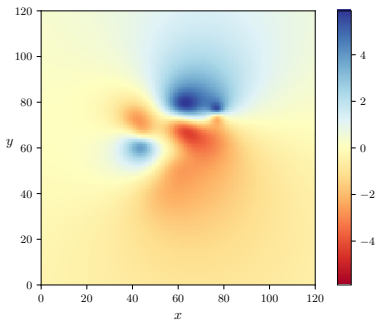
(i) Velocity Magnitude: Neural Network $t = 1s$.



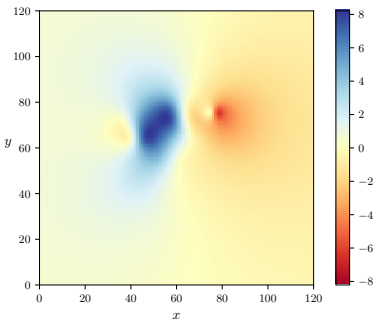
(j) Error map: $t = 1s$.

Figure 4.2: Comparison of the velocity field predicted by *Vortex-Network* at $t = 1s$ with the actual velocity field from numerical simulation

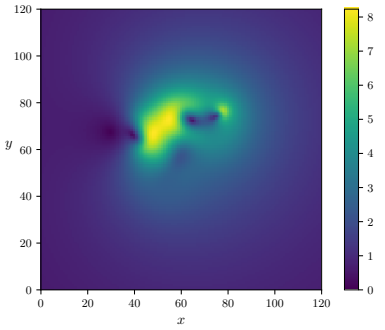
https://doi.org/10.24355/dbbs.084-202103121124-0



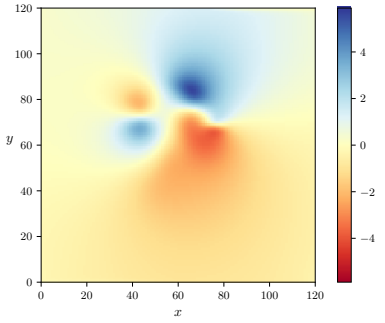
(a) Velocity (x-component) $t = 0s$.



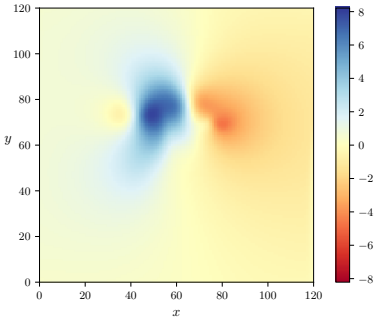
(b) Velocity (y-component) $t = 0s$.



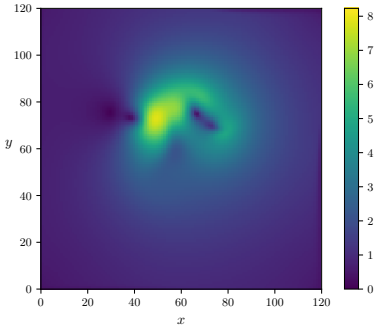
(c) Velocity Magnitude $t = 0s$.



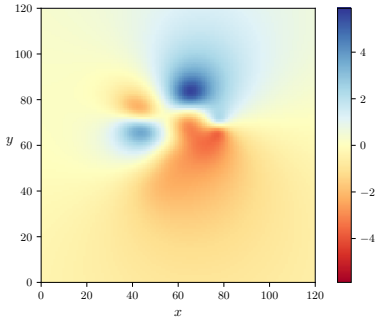
(d) Velocity (x-component): Simulation $t = 2s$.



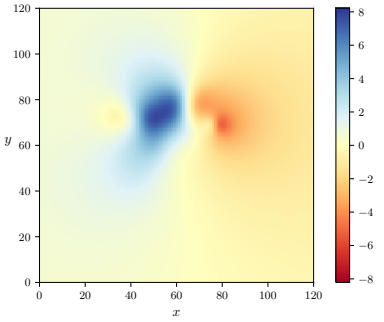
(e) Velocity (y-component): Simulation $t = 2s$.



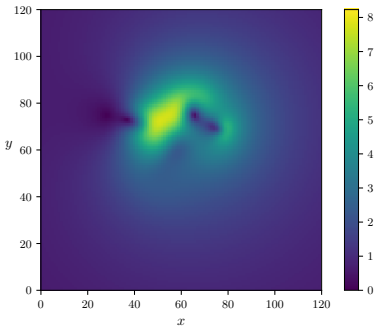
(f) Velocity Magnitude: Simulation $t = 2s$.



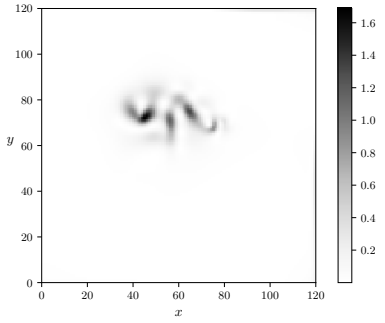
(g) Velocity (x-component): Neural Network $t = 2s$.



(h) Velocity (y-component): Neural Network $t = 2s$.



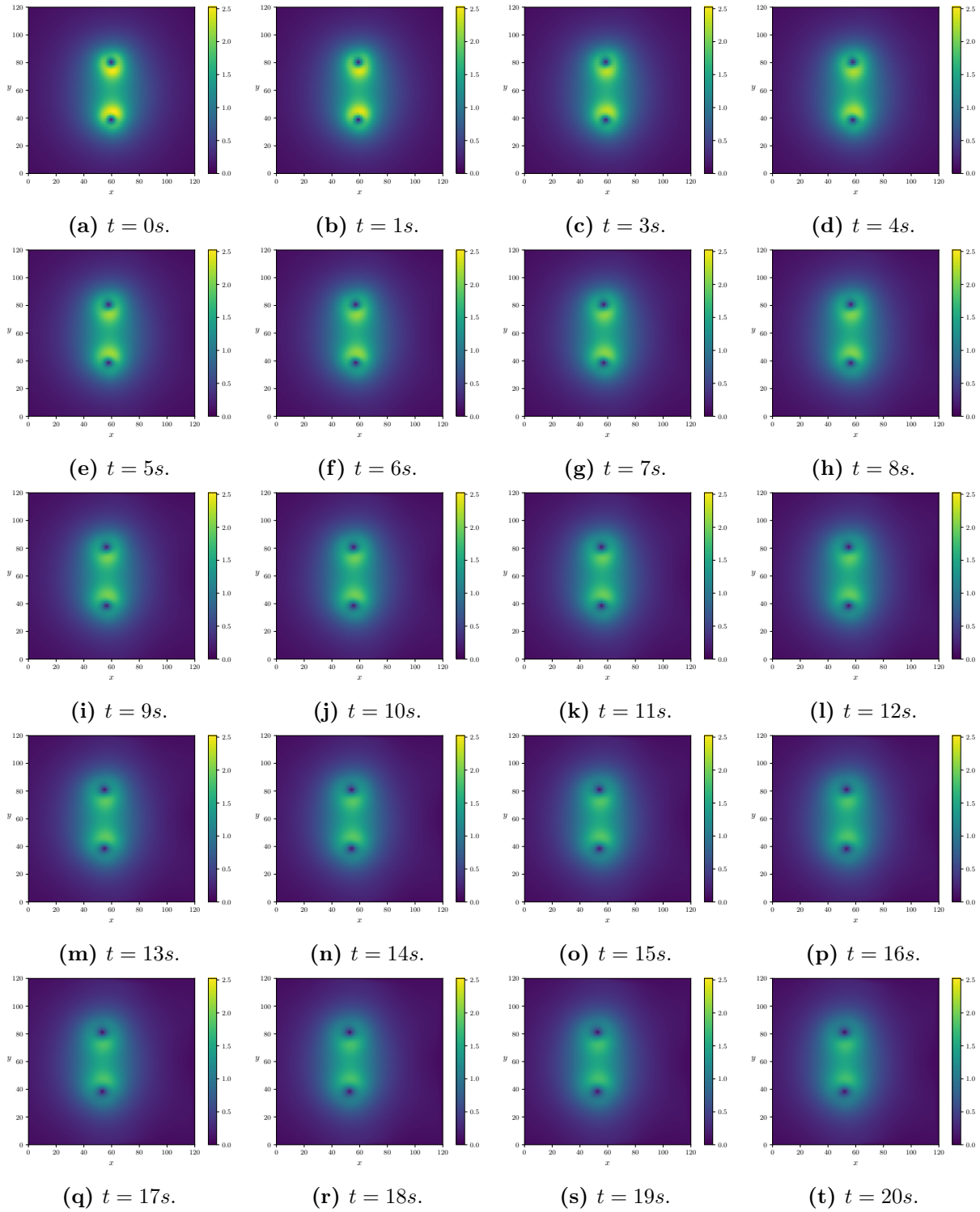
(i) Velocity Magnitude: Neural Network $t = 2s$.



(j) Error map: $t = 2s$.

Figure 4.3: Comparison of the velocity field predicted by *Vortex-Network* at $t = 2s$ with the actual velocity field from numerical simulation

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) $t = 0s$. **(b)** $t = 1s$. **(c)** $t = 3s$. **(d)** $t = 4s$.

(e) $t = 5s$. **(f)** $t = 6s$. **(g)** $t = 7s$. **(h)** $t = 8s$.

(i) $t = 9s$. **(j)** $t = 10s$. **(k)** $t = 11s$. **(l)** $t = 12s$.

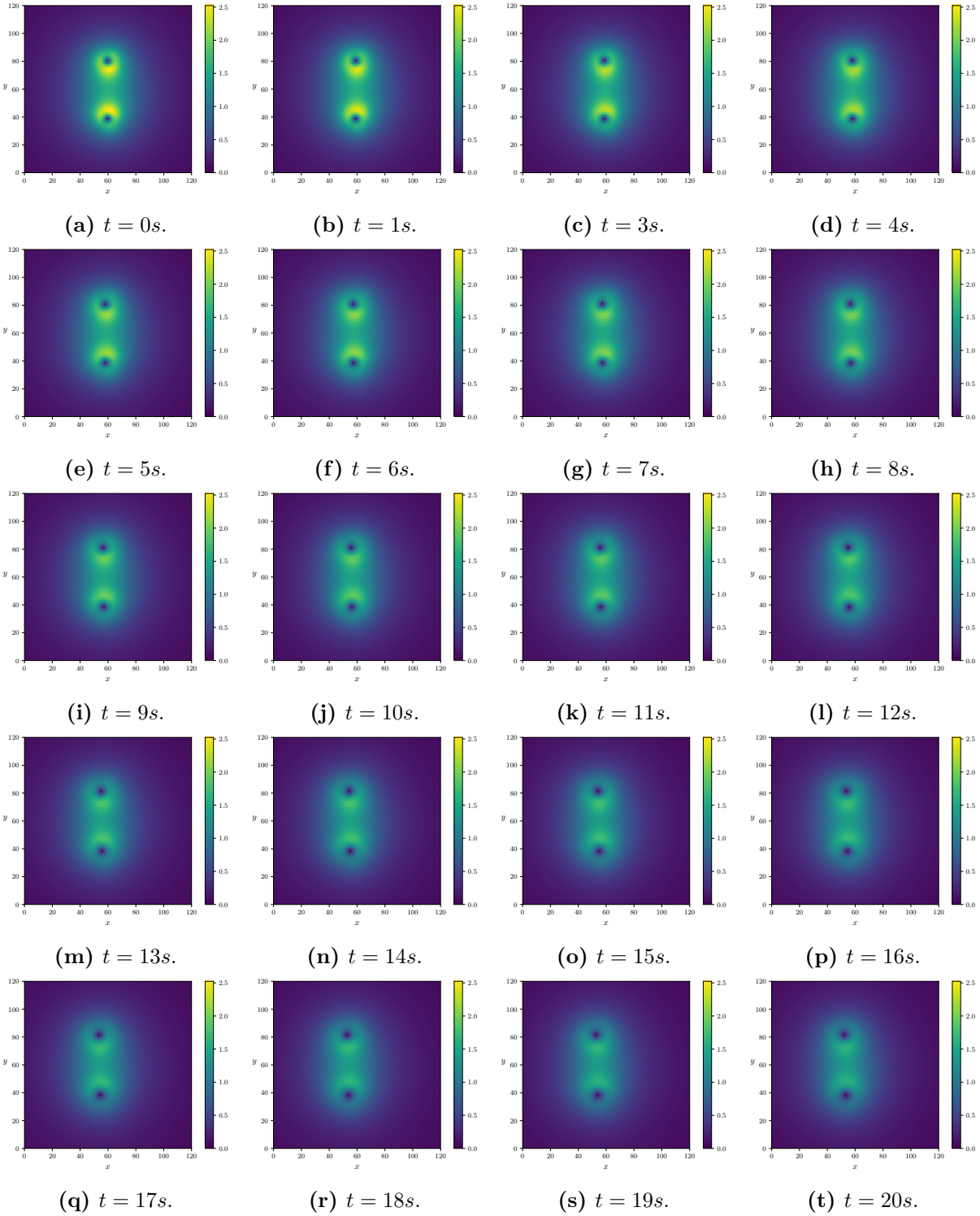
(m) $t = 13s$. **(n)** $t = 14s$. **(o)** $t = 15s$. **(p)** $t = 16s$.

(q) $t = 17s$. **(r)** $t = 18s$. **(s)** $t = 19s$. **(t)** $t = 20s$.

Figure 4.4: Velocity magnitudes of 2 similar counter-rotating vortices obtained from numerical simulation

show an unusual behaviour of the curve obtained by neural network as compared to the *Vortex-Fit one*. However, this is not an issue, since theoretically there should be no vertical movement. The curve from *Vortex-Fit* shows some movement in y due to errors associated with numerical simulations. Therefore, our neural network also predicts some movement in y. However, if we look carefully, this movement in y is negligible as compared to their counterparts in x. Over a time of $20s$, the particles in this case moves by upto 7 grid cell lengths along x, whereas the

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) $t = 0s$. **(b)** $t = 1s$. **(c)** $t = 3s$. **(d)** $t = 4s$.

(e) $t = 5s$. **(f)** $t = 6s$. **(g)** $t = 7s$. **(h)** $t = 8s$.

(i) $t = 9s$. **(j)** $t = 10s$. **(k)** $t = 11s$. **(l)** $t = 12s$.

(m) $t = 13s$. **(n)** $t = 14s$. **(o)** $t = 15s$. **(p)** $t = 16s$.

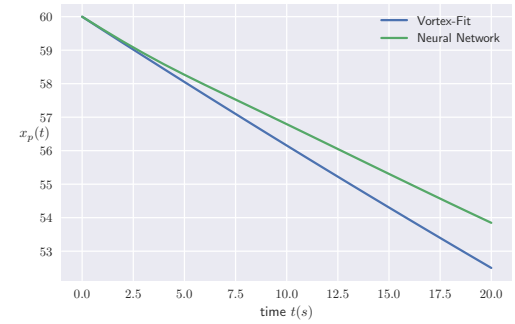
(q) $t = 17s$. **(r)** $t = 18s$. **(s)** $t = 19s$. **(t)** $t = 20s$.

Figure 4.5: Velocity magnitudes of 2 similar counter-rotating vortices obtained by predictions from *Vortex-Network*

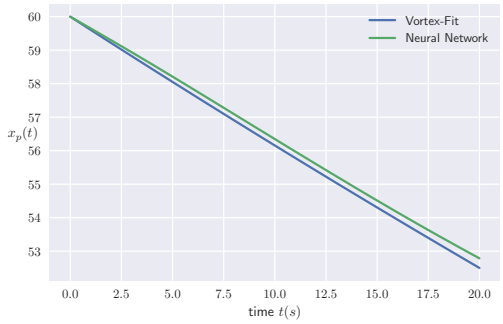
movement in y is less than half a grid cell length over the same time span.

Similar to the previous example, we show an another example with 2 vortex particles, again located along the same vertical line, but this time have strengths of same magnitude and sign. Thus, we now have vortices rotating in a similar manner. Dynamics of such a system of particles results in a rotational motion of both the particles about the centre of the line joining these two particles. This is evident from Fig. 4.7, which shows the velocity magnitudes obtained by

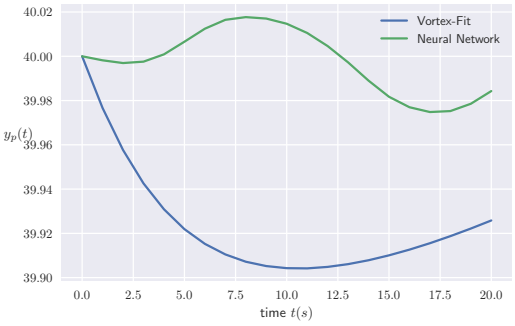
https://doi.org/10.24355/dbbs.084-202103121124-0



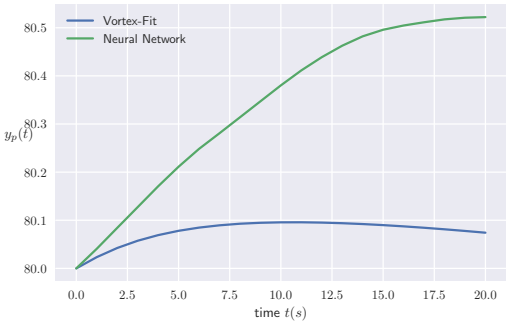
(a) Particle 1: $x_p(t)$.



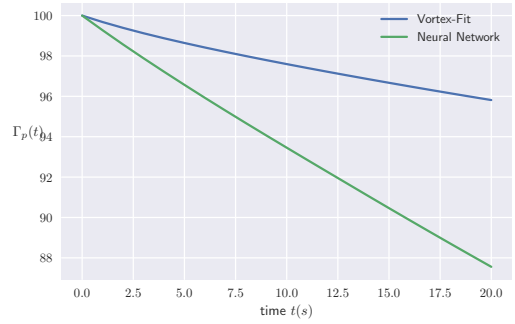
(b) Particle 2: $x_p(t)$.



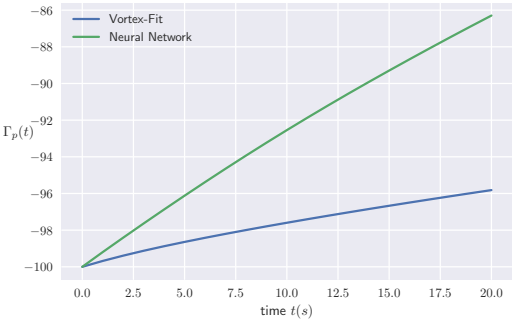
(c) Particle 1: $y_p(t)$.



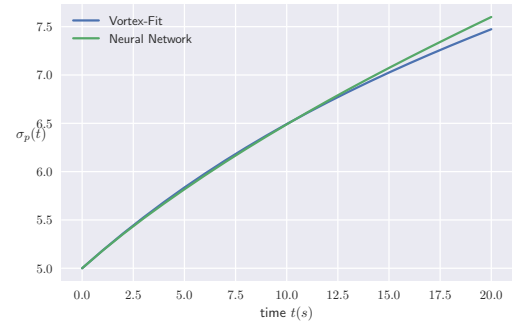
(d) Particle 2: $y_p(t)$.



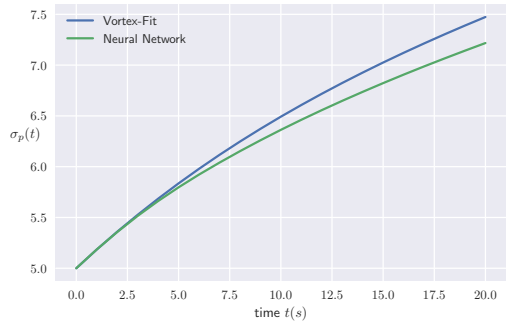
(e) Particle 1: $\Gamma_p(t)$.



(f) Particle 2: $\Gamma_p(t)$.



(g) Particle 1: $\sigma_p(t)$.

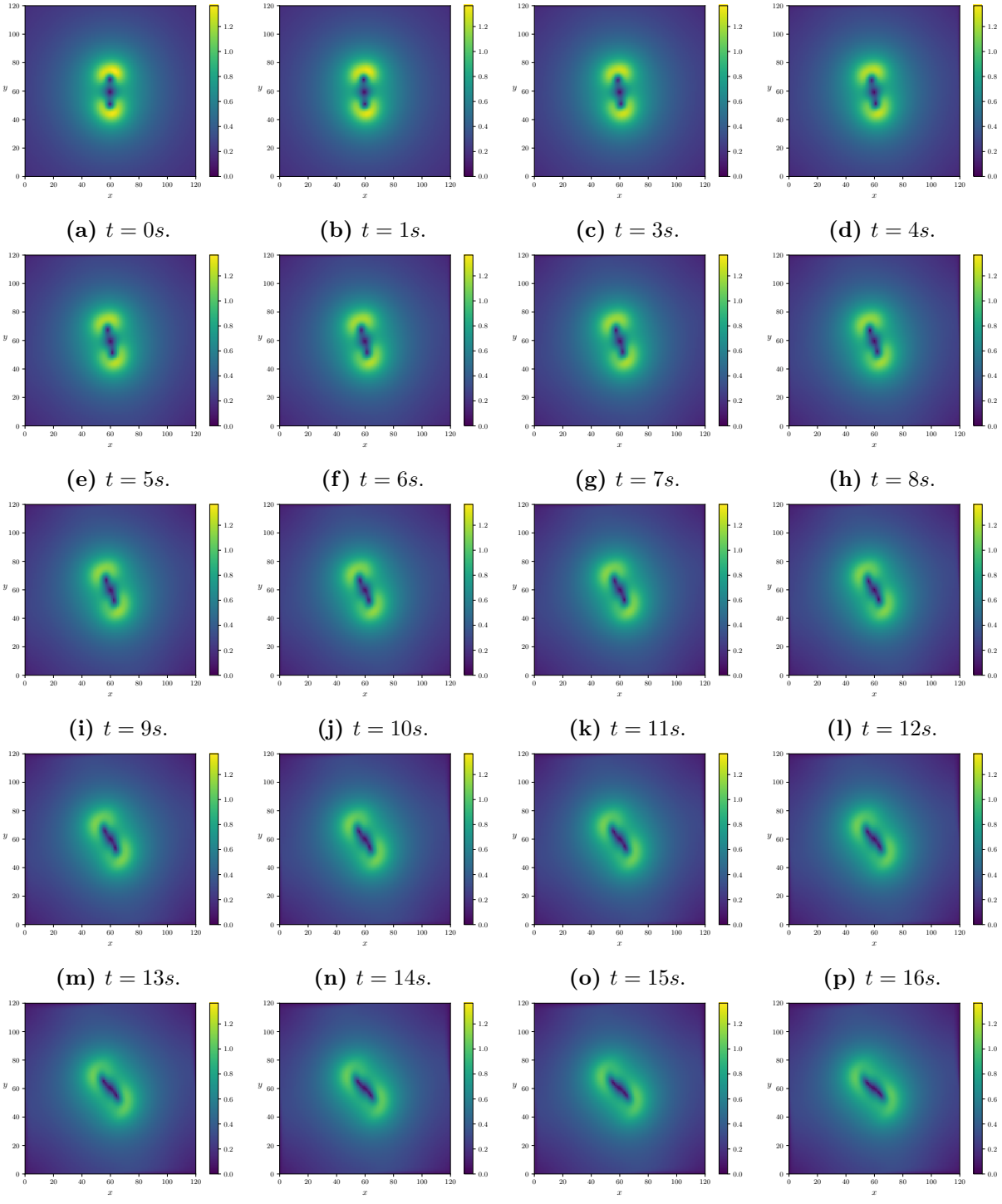


(h) Particle 2: $\sigma_p(t)$.

Figure 4.6: Comparison of evolution curves for 2 similar counter-rotating vortices obtained by predictions from *Vortex-Network* with *Vortex-Fit*

numerical simulation. The interesting part is our neural network also emulates this behaviour of rotational motion, as shown in Fig. 4.8.

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) $t = 0s$. **(b)** $t = 1s$. **(c)** $t = 3s$. **(d)** $t = 4s$.

(e) $t = 5s$. **(f)** $t = 6s$. **(g)** $t = 7s$. **(h)** $t = 8s$.

(i) $t = 9s$. **(j)** $t = 10s$. **(k)** $t = 11s$. **(l)** $t = 12s$.

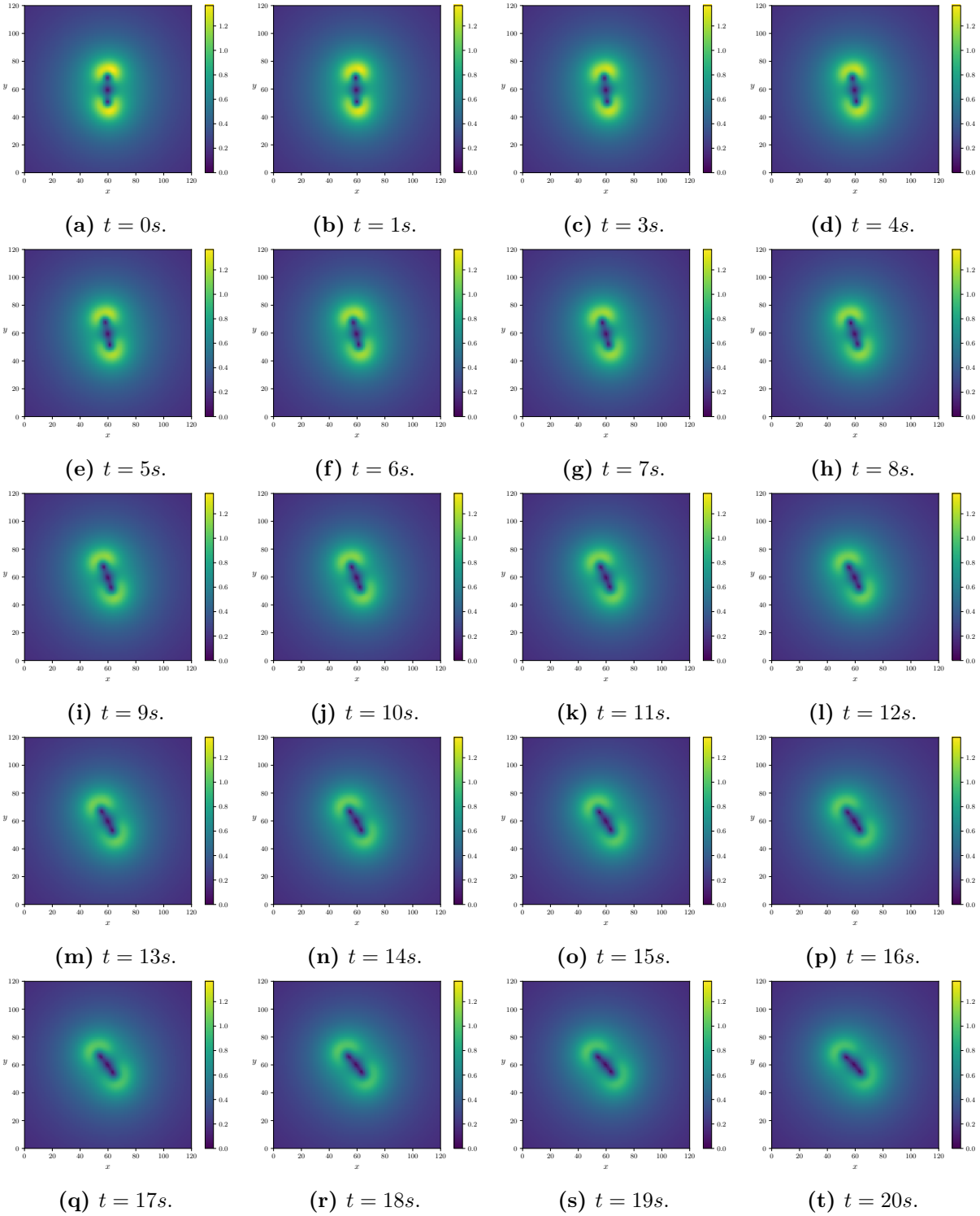
(m) $t = 13s$. **(n)** $t = 14s$. **(o)** $t = 15s$. **(p)** $t = 16s$.

(q) $t = 17s$. **(r)** $t = 18s$. **(s)** $t = 19s$. **(t)** $t = 20s$.

Figure 4.7: Velocity magnitudes of 2 similar rotating vortices obtained from numerical simulation

Since both particles rotate about the mid-point of the line joining them, the evolution curves for their positions should be sinusoidal in nature. We observe exactly such a behaviour both from the *Vortex-Fit* and from our *Vortex-Network* predictions, as shown in Fig. 4.9. Unlike the case with 2 counter-rotating vortices, we do not see an unusual behaviour in the y-movement of the particles. Here, physically the movement along both x and y are equally significant, and we see a very strong resemblance with *Vortex-Fit* to the predictions of both of these movements.

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) $t = 0s$. **(b)** $t = 1s$. **(c)** $t = 3s$. **(d)** $t = 4s$.

(e) $t = 5s$. **(f)** $t = 6s$. **(g)** $t = 7s$. **(h)** $t = 8s$.

(i) $t = 9s$. **(j)** $t = 10s$. **(k)** $t = 11s$. **(l)** $t = 12s$.

(m) $t = 13s$. **(n)** $t = 14s$. **(o)** $t = 15s$. **(p)** $t = 16s$.

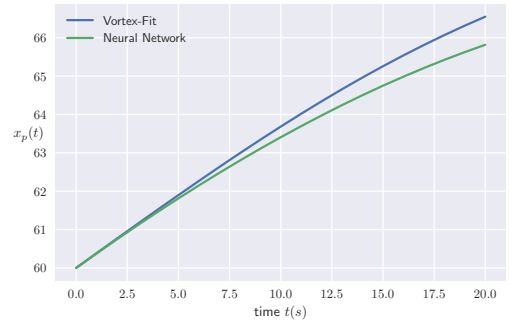
(q) $t = 17s$. **(r)** $t = 18s$. **(s)** $t = 19s$. **(t)** $t = 20s$.

Figure 4.8: Velocity magnitudes of 2 similar rotating vortices obtained by predictions from *Vortex-Network*

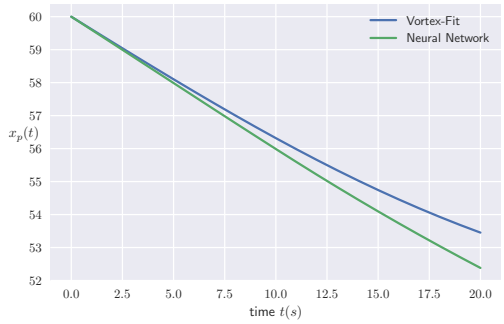
The examples presented here shows that our *Vortex-Network* is capable of delivering predictions which obeys some of the fundamental intuitions associated with the dynamics of such simple 2 particle system.

Now we will consider an example for flows in presence of boundaries. We combine the vortex predictions dynamics from the *Vortex-Network-2* and **BC-Net** to obtain the predictions of particle dynamics followed by computation of an appropriate correction velocity field. Fig. 4.11 and

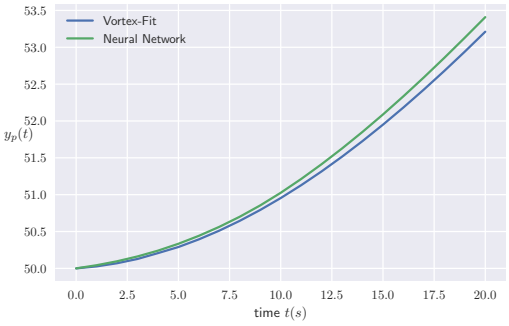
https://doi.org/10.24355/dbbs.084-202103121124-0



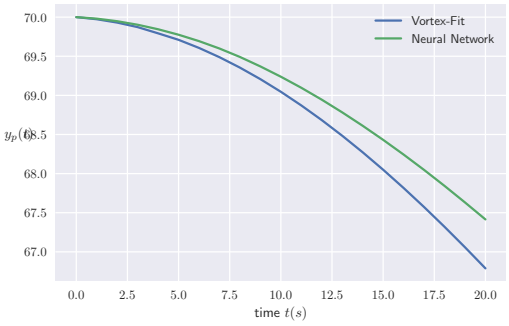
(a) Particle 1: $x_p(t)$.



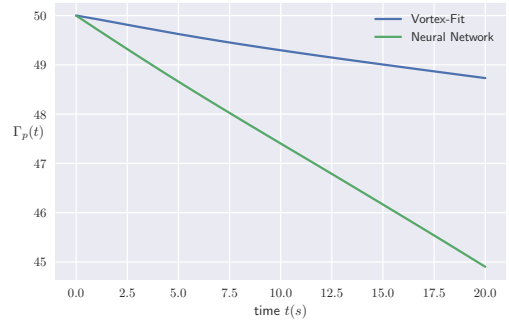
(b) Particle 2: $x_p(t)$.



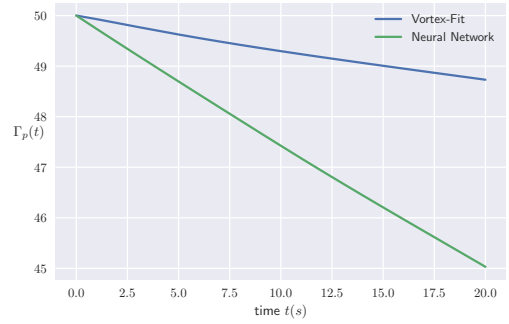
(c) Particle 1: $y_p(t)$.



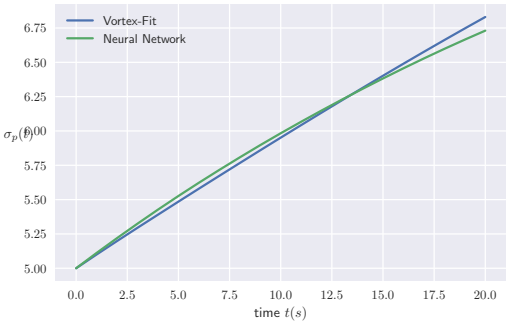
(d) Particle 3: $y_p(t)$.



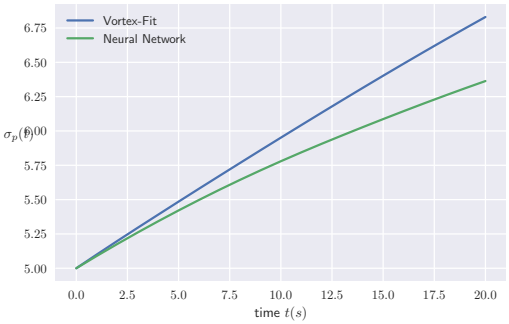
(e) Particle 1: $\Gamma_p(t)$.



(f) Particle 2: $\Gamma_p(t)$.



(g) Particle 1: $\sigma_p(t)$.



(h) Particle 2: $\sigma_p(t)$.

Figure 4.9: Comparison of evolution curves for 2 similar rotating vortices obtained by predictions from *Vortex-Network* with *Vortex-Fit*

Fig. 4.12 shows an example of the outcome of both the networks in action. The velocity field obtained by the *Vortex-Network* does not naturally satisfy the no-through-flow condition at the boundaries. This is evident from Fig. 4.11c, where the x-component of velocity does not neces-

https://doi.org/10.24355/dbbs.084-202103121124-0

sarily vanish at the vertical boundaries. This is exactly the case for the y-component of velocity at the horizontal boundaries, as shown in Fig. 4.12c. Addition of the correction field delivered by *BC-Net* reduces the error corresponding to the square of velocity magnitudes summed over all the grid cells from 1289.43 to 402.82 for the example shown in Fig. 4.11c and Fig. 4.12c. Also the boundary condition loss decreases from 24.24 to 0.032. This could also be visualised from Fig. 4.13, where we plot the normal component of velocities at all the four boundaries before and after the application of *BC-net*. There is another constraint during training of *BC-net* that the correction field produced is divergence-free in order to make the overall velocity field divergence-free, since the velocity field obtained by *Vortex-Network* is divergence-free by default. Fig. 4.10 shows the divergence map of the correction field, where the divergence is computed at locations corresponding to the centers of the grid cells. It could be seen from Fig. 4.10 that the divergence is almost negligible in most of the domain. These results demonstrates the impact of our *BC-Net* to satisfy the no-through-flow boundary condition, while also the physics of incompressibility constraint.

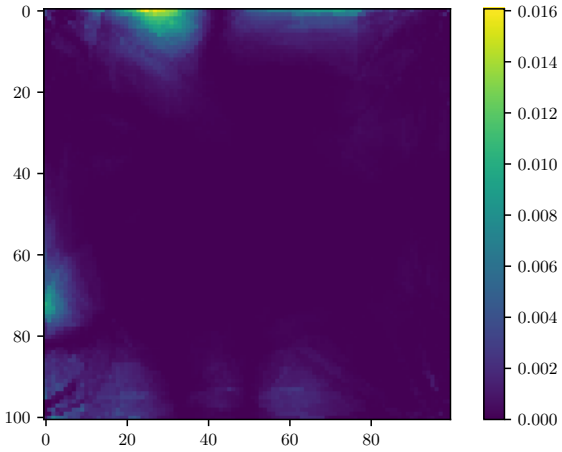
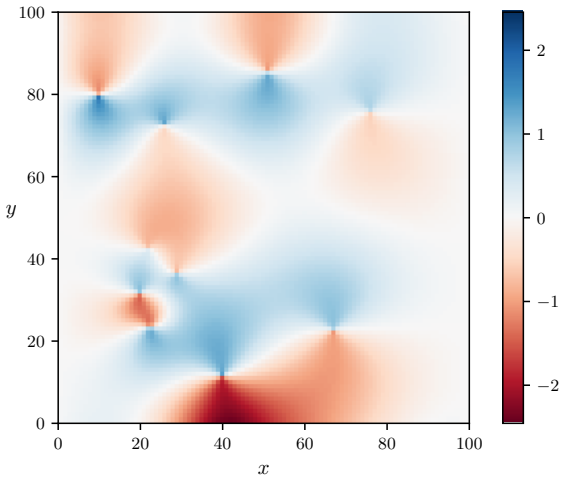
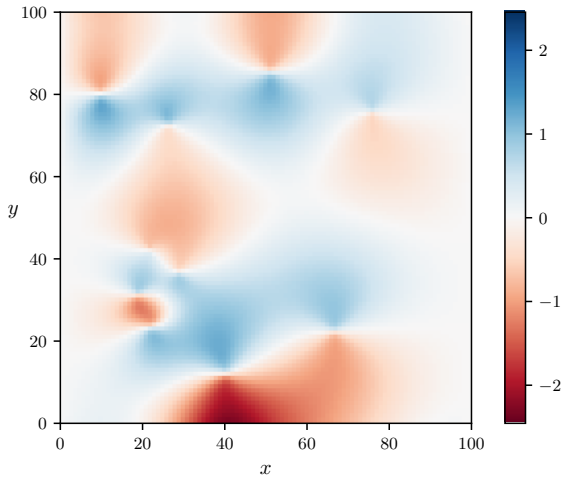


Figure 4.10: Divergence map of the correction field delivered by *BC-net*

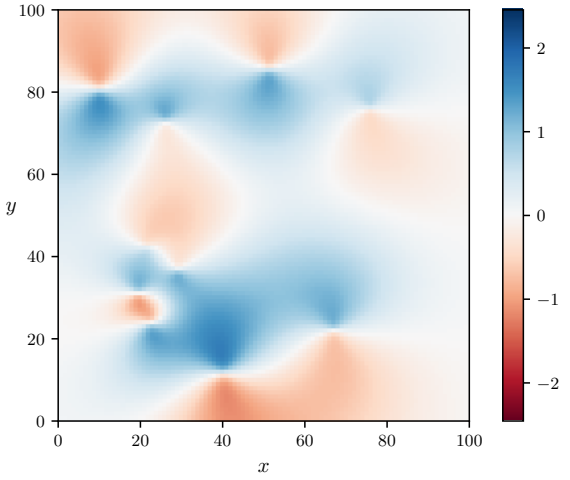
https://doi.org/10.24355/dbbs.084-202103121124-0



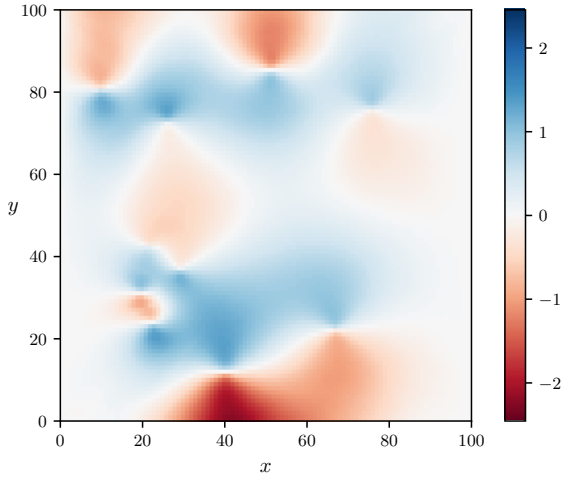
(a) x-velocity: $t = 0s$.



(b) x-velocity: Simulation $t = 1s$.



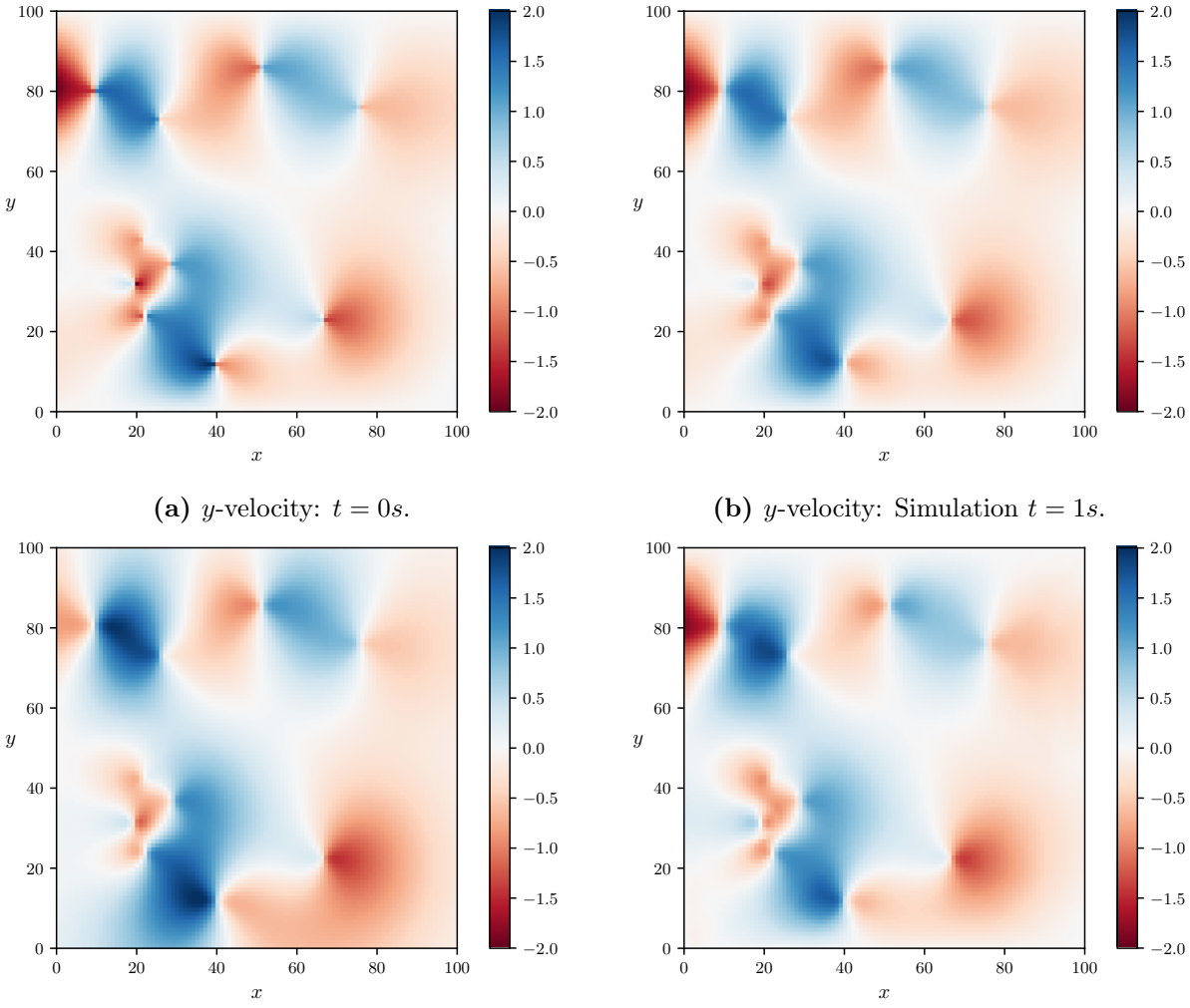
(c) x-velocity: *Vortex-Net* $t = 1s$.



(d) x-velocity: *Vortex-Net*+*BC-Net* $t = 1s$.

Figure 4.11: Comparison of the x-component of the velocity field before and after the application of *BC-net*

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) y-velocity: $t = 0s$.

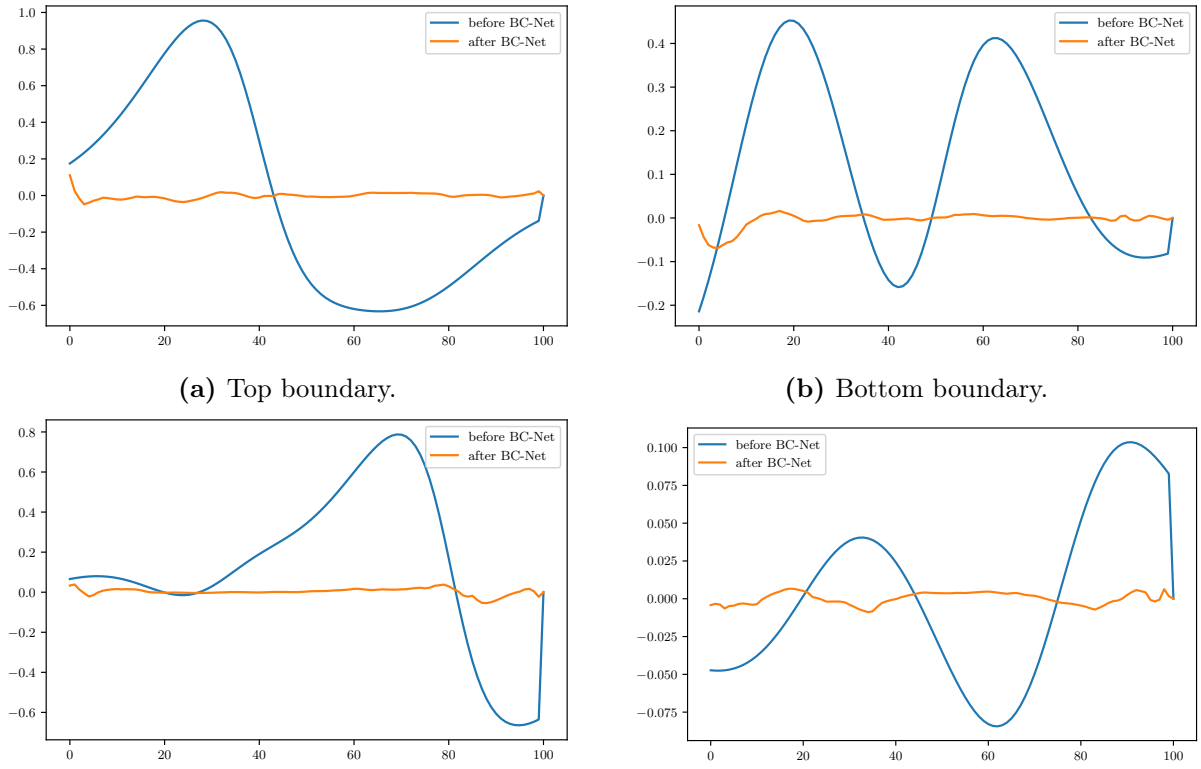
(b) y-velocity: Simulation $t = 1s$.

(c) y-velocity: *Vortex-Net* $t = 1s$.

(d) y-velocity: *Vortex-Net+BC-Net* $t = 1s$.

Figure 4.12: Comparison of the y-component of the velocity field before and after the application of *BC-net*

https://doi.org/10.24355/dbbs.084-202103121124-0



(a) Top boundary.

(b) Bottom boundary.

(c) Left boundary.

(d) Right boundary.

Figure 4.13: Comparison of the normal component of the velocity at the boundaries before and after the application of *BC-net*

https://doi.org/10.24355/dbbs.084-202103121124-0

Chapter 5

Summary and Conclusions

In this work, we have presented and demonstrated the applicability of our proposed approaches for learning Lagrangian vortex particle dynamics using deep neural networks. We began with the discussion of the basic governing equations of fluid flow in the form of incompressible Navier-Stoke equations. It was followed by the discussion of the Eulerian and Lagrangian description of flow, with their general advantages and disadvantages. Then we discussed about some of the common steps involved in the numerical solvers based on the Eulerian description of flow, and in turn pointed out some of the key aspects related to the high computational complexity of the *pressure solve* step. We followed it with the description of particle based methods based on the Lagrangian description of the flow. We again pointed out some of the key aspects, especially the disadvantages associated with the Smoothed Particle Hydrodynamics approach. We then moved further with a much detailed and elaborate description of Vortex Particle methods. One key point here to note was that for flows in open domain, we always obtain a divergence-free velocity field from the solution of vortex methods. Also, the number of particles required to describe the flow would be less as compared to the SPH approaches, since the particles need to be present only at the regions of vorticity. In our work, we consider the flow field to be parametrized by discrete number of Lagrangian vortex particles. We have a parametrization identical to the vortex methods with gaussian vortex blobs as the fundamental computational elements. We consider a discrete set of vortex particles, with each particle having their own particle strength and a core size. Such set of vortex particles describe a flow field whose dynamics is of utmost interest in this work. We set up with task of dealing with the dynamics of the flow field in terms of the dynamics of the discrete set of vortex particles, with evolution of their positions, strengths and core sizes are of utmost interest.

The objective to successfully train and evaluate the neural networks to learn and predict the dynamic behaviour of the vortex particles has been achieved both for an open domain and for flows in presence of solid boundaries. The major part of this work is based on a new approach to model interaction between particles inside the neural network. In previous dealing with particle dynamics in the context of deep learning, in general graph neural networks are typically used to model such inter-particle interactions. However, such networks are computationally too expensive, since it is required to have $O(N_p^2)$ executions of the neural neural in order to predict the dynamics over a single time step. We then presented our approach to model particle interactions which is based on the Taylor series approximation of the influence field. We argue and demonstrate with our results that with better and better approximation of the influence field using higher order influence vectors, performance of our *Vortex-Networks* gets closer and closer to the performance of a full *Interaction-Network*. Also, our approach only requires $O(N_p)$ executions of the neural neural in order to predict the dynamics over a single time step. However, this complexity could further be reduced, if one considers the interaction of any particle with only a specific set of particles nearby it, instead of all the other particles. This would be a

https://doi.org/10.24355/dbbs.084-202103121124-0

reasonable approximation, since we use a gaussian falloff-kernel to model the vorticity distribution around the particles, and therefore the influence of any particle on any other particle would be decreasing as the distance between them is higher and higher. Even though we experiment using different orders of influence vectors on a single dataset generated for experimentation purposes, the actual order of the influence vector to use in any given application would largely depend on the complexity of the flow field involved. We do not provide any specific estimate or relationship for the type of the order of the influence vector to be used based on the description of velocity field. It is also important to note that our approach of using data samples in the form of grid based velocity fields to learn for the dynamics of particles in terms of vorticity work only for flows in an open domain.

The visual demonstration of the simulations using our proposed *Vortex-Network* in action has shown that the proposed approach predicts the dynamic behaviour of the vortex particles with a reasonable accuracy. However, we face the same problem, as one typically encounters in any data-driven approaches. The results from the neural network predictions might not exactly match the true dynamics of the system. Also, the predictions form neural networks are typically non-convincing or worse for data points that are far out of the training distribution. Especially in the areas where the problem under concern is based on fundamental physics, like in our case with fluid dynamics, it is important to have a reasoning for the predictions from neural networks for a specific input. In case of classical numerical methods, such causal relationships are easy to obtain, whereas for neural networks, obtaining such causal relationships is difficult and is an important topic of research in the machine learning community. However, the particle-based deep learning approaches, especially the approach based on vortex particles would fare much better in understanding the cause-effect relationship as compared to their grid-based counterparts. In particle-based methods, neural networks learns to base their predictions on some set of discrete particle features and predicts the evolution of fluid particles, which are actual physical quantities, whereas deep learning approaches based on the grid-based representation of the fluid, lays the foundation for its learning on values associated with some random grid locations. Such lack of the establishment of causal relationships and lack of accuracy as compared to high resolution numerical simulations prevents application of works like ours in safety critical applications like aerodynamics. However, for applications like computer graphics, the accuracy is usually not paramount, rather the visual features created by the simulators is vital. In such scenarios, deep-learning-based solvers could provide real-time animations.

Another, important issue that we had to address in our work is the presence of solid boundaries. We have discussed the difficulties associated when dealing with such boundary conditions in terms of vorticity. Therefore, we had to take a different path by learning a correction field in terms of velocity, instead of vorticity, using a separate neural network. Even from the results, it was evident that the delivered correction field provides much better improvement on the obtained flow fields as compared to the one without such a correction field. However, we lose a very important property that we had for open domain scenarios, the flow fields were always divergence-free. With the inclusion of the boundary condition network, this is no more a given. Rather, we had to enforce the satisfaction of such incompressibility constraint in terms of loss function during training. This is only a weak way of achieving the property being satisfied. This would not have been the case, if we were to deal with boundaries also in terms o vortex particles. A natural continuation of this work would be to develop mechanism to learn for particle dynamics in terms of vorticity itself as the primary variable. Also, many of the interesting flow patterns occur in presence of obstacles and boundaries of complicated shapes. Since our approach works well for an open domain scenario, it is rather quite inferior yet in terms of its ability to deal with much more complicated and intricate flows.

https://doi.org/10.24355/dbbs.084-202103121124-0

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI) 16)*, pages 265–283, 2016.

[2] Christopher Anderson and Claude Greengard. On vortex methods. *SIAM journal on numerical analysis*, 22(3):413–440, 1985.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[5] Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *arXiv preprint arXiv:1612.00222*, 2016.

[6] J Thomas Beale and Andrew Majda. High order accurate vortex methods with explicit velocity kernels. *Journal of Computational Physics*, 58(2):188–208, 1985.

[7] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 2015 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 2015.

[8] G Brenner. Cfd in process engineering. In *100 Volumes of âNotes on Numerical Fluid Mechanicsâ*, pages 351–359. Springer, 2009.

[9] Robert Bridson. *Fluid simulation for computer graphics*. CRC press, 2015.

[10] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[11] Alexandre Joel Chorin. The numerical solution of the navier-stokes equations for an incompressible fluid. *Bulletin of the American Mathematical Society*, 73(6):928–931, 1967.

[12] Alexandre Joel Chorin et al. Numerical study of slightly viscous flow. *J. Fluid Mech*, 57(4):785–796, 1973.

[13] Georges-Henri Cottet, Petros D Koumoutsakos, et al. *Vortex methods: theory and practice*, volume 8. Cambridge university press Cambridge, 2000.

[14] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, 20(1):30–42, 2011.

[15] Marie Dillon Dahleh. Analysis and application of the discrete vortex method. 1991.

https://doi.org/10.24355/dbbs.084-202103121124-0

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[17] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical models and image processing*, 58(5):471–483, 1996.

[18] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[19] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] H von Helmholtz. Über integrale der hydrodynamischen gleichungen, welche den wirbelbewegungen entsprechen. *Journal für die reine und angewandte Mathematik*, 1858(55):25–55, 1858.

[23] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14:8, 2012.

[24] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *arXiv preprint arXiv:2001.07457*, 2020.

[25] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[26] Antony Jameson. A perspective on computational algorithms for aerodynamic analysis and design. *Progress in Aerospace Sciences*, 37(2):197–243, 2001.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Petros Koumoutsakos. Multiscale flow simulations using particles. *Annu. Rev. Fluid Mech.*, 37:457–487, 2005.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[30] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[31] Anthony Leonard. Vortex methods for flow simulation. *Journal of Computational Physics*, 37(3):289–335, 1980.

[32] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on χ-transformed points. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 828–838, 2018.

https://doi.org/10.24355/dbbs.084-202103121124-0

[33] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. pages 3431–3440, 2015.

[34] Joseph J Monaghan. An introduction to sph. *Comput. Phys. Comm.*, 48:89–96, 1988.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[36] Mirta Perlman. On the accuracy of vortex methods. *Journal of Computational Physics*, 59(2):200–223, 1985.

[37] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[38] Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

[39] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[40] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[41] Pierre-Arnaud Raviart. An analysis of particle methods. In *Numerical methods in fluid dynamics*, pages 243–324. Springer, 1985.

[42] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

[43] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[44] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018.

[45] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. In *ACM SIGGRAPH 2005 Papers*, pages 910–914. 2005.

[46] A Senior, V Vanhoucke, P Nguyen, T Sainath, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 2012.

[47] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999.

[48] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27:3104–3112, 2014.

[49] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6411–6420, 2019.

[50] Shiying Xiong, Xingzhe He, Yunjin Tong, and Bo Zhu. Neural vortex method: from finite lagrangian particles to infinite dimensional eulerian dynamics. *arXiv preprint arXiv:2006.04178*, 2020.

https://doi.org/10.24355/dbbs.084-202103121124-0

List of Figures

https://doi.org/10.24355/dbbs.084-202103121124-0

https://doi.org/10.24355/dbbs.084-202103121124-0

List of Tables

https://doi.org/10.24355/dbbs.084-202103121124-0